
1.1. Lenguaje de Bajo Nivel.

Se denomina *lenguaje máquina* a la serie de datos que la parte física de la computadora o hardware, es capaz de interpretar.

Una computadora digital o, mejor dicho, su parte física, sólo distingue datos de tipo binario, es decir, constituidos por dos únicos valores a los que se denomina valor 0 y valor 1 y que, físicamente, se materializan con tensiones comprendidas entre 0 y 4.0 voltios y entre 4 y 5 voltios, respectivamente. Para representar datos que contengan una información se utilizan una serie de unos y ceros cuyo conjunto indica dicha información.

La información que hace que el hardware de la computadora realice una determinada actividad se llama *instrucción*. Por consiguiente una instrucción es un conjunto de unos y ceros. Las instrucciones así formadas equivalen a acciones elementales de la máquina, por lo que al conjunto de dichas instrucciones que son interpretadas directamente por la máquina se denomina *lenguaje máquina*.

El lenguaje máquina fue el primero que empleo el hombre para la programación de las primeras computadoras. Una instrucción en lenguaje máquina puede representarse de la siguiente forma:

011011001010010011110110

Esta secuencia es fácilmente ejecutada por la computadora, pero es de difícil interpretación, siendo aun mas difícil la interpretación de un programa (conjunto de instrucciones) escrito de esta forma. Esta dificultad hace que los errores sean frecuentes y la corrección de los mismos costosa, cuando no imposible, al igual que la verificación y modificación de los programas.

La anterior secuencia de dígitos binarios (bits) puede indicar a la computadora que:

<<Traslade el contenido de la posición de memoria X a la posición de memoria Y.>>

Si lo vemos escrito de esta forma, lo entenderemos fácilmente, ya que está en nuestro lenguaje natural, pero la máquina elemental será incapaz de entender nada. Vemos, pues, que la forma de indicar a la máquina lo que debe hacer es totalmente diferente de la indicar a un ser humano lo mismo, por lo que deben emplearse sistemas de traducción de una forma a otra.

Ya se ha dicho que en un principio el programador empleaba directamente el lenguaje máquina. En este caso el traductor era el programador; pero vimos también los problemas que esto causaba.

Con la práctica en el manejo de la máquina se cayó en la cuenta de que se podría utilizar la propia máquina para ayudar en la traducción de estos programas. Es decir, que si a una máquina elemental se le dotaba de un programa, también elemental, que tradujera un número determinado de caracteres alfabéticos en una secuencia de unos y ceros, se podría escribir un programa constituido por una secuencia de grupos de caracteres alfabéticos, en la que cada uno de los grupos indicaría una acción a realizar por el ordenador y, una vez escrito el programa, sería la propia máquina la que pasaría los grupos de caracteres a bits.

Las ventajas de esto son evidentes, ya que para el hombre resulta mas fácil manipular grupos de caracteres y la traducción se haría de manera automática. Por ejemplo, se podría escribir:

TRASLADAR 11010110, 00011101

Esto indicaría que el contenido de la posición 11010110 había que pasarlo a la posición 00011101 si se sabe que al grupo alfabético TRASLADAR le corresponde la secuencia de bits 11110101. La máquina traduciría la anterior instrucción como:

11110101 11010110 00011101

Al grupo alfabético se le denomina *mnemotécnico*, y existirá un mnemotécnico por cada instrucción. Se le da este nombre porque sirve para recordar con mayor facilidad el conjunto de instrucciones de una determinada máquina.

De esta forma aparecieron los lenguajes ensambladores (Assembler, en inglés). Poco a poco, con el avance de la programación (Software), estas primeras y sencillas ayudas se fueron haciendo más complejas, permitiendo que, además de los mnemotécnicos correspondientes a la operación a realizar, se pudieran emplear otros para indicar, por ejemplo, los operandos. La anterior instrucción se podría escribir de la siguiente forma:

TRASLADAR POS-A POS-B

Que nos resulta de más fácil comprensión.

También se introdujo la posibilidad de indicar a la computadora la dirección de un salto en la secuencia de ejecución de un programa mediante la utilización de etiquetas.

A los programas que permiten pasar del programa escrito de esta manera (programa fuente, en ensamblador) al *lenguaje máquina* también se les denomina normalmente *ensambladores*. Estos traductores, como ya se ha dicho, se fueron complicando cada vez más para que la labor del programador fuera más fácil, incluyendo los denominados directivos del ensamblador, que son órdenes o informaciones que el programador da al traductor, no instrucciones de lenguaje máquina.

Aun con todas estas sofisticaciones y ayudas, el programador de lenguaje ensamblador debe conocer perfectamente el sistema físico (Hardware) de la máquina con que trabaja, pues aunque emplee mnemotécnicos, etiquetas, etc., éstas sirven para indicar una posición de memoria determinada, un registro o cualquier otra parte de la máquina.

Por eso se dice que el *lenguaje ensamblador* es un *lenguaje de bajo nivel*, es decir, ligado con el <<hard>> concreto de una determinada máquina. Aquí radica la diferencia importante con los lenguajes más complejos, llamados de alto nivel, como el Basic, Pascal, Cobol, etc., ya que en éstos el programador no tiene porque reconocer el <<hard>> de la máquina. Trabaja con variables, constantes e instrucciones simbólicas, y es el traductor quien las transforma en las direcciones apropiadas.

1.2. VENTAJAS DE LOS LENGUAJES ENSAMBLADORES

El corazón de la computadora es el microprocesador, éste maneja las necesidades aritméticas, de lógica y de control de la computadora.

El microprocesador tiene su origen en la década de los sesenta, cuando se diseñó el circuito integrado (IC por sus siglas en inglés) al combinar varios componentes electrónicos en un solo componente sobre un "chip" de silicio.

Los fabricantes colocaron este diminuto chip en un dispositivo parecido a un ciempiés y lo conectaron a un sistema en funcionamiento. A principios de los años setenta Intel introdujo el chip 8008 el cual, instalado en una computadora terminal, acompañó a la primera generación de microprocesadores.

En 1974 el 8008 evolucionó en el 8080, un popular microprocesador de la segunda generación para propósitos generales. En 1978 Intel produjo la tercera generación de procesadores 8086, para proporcionar alguna compatibilidad con el 8080 y que representan un avance significativo de diseño.

Después, Intel desarrolló una variación del 8086 para ofrecer un diseño sencillo y compatibilidad con los dispositivos de entrada/salida de ese momento. Este nuevo procesador, el 8088, fue seleccionado por IBM para su computadora personal en 1981. Una versión mejorada del 8088 es el 80188, y versiones mejoradas del 8086, son los 80186, 80286, 80386, 80486 y el Pentium, cada uno de ellos permite operaciones adicionales y más procesamiento.

La variedad de microcomputadoras también ocasionó un renovado interés en el lenguaje ensamblado, cuyo uso conlleva a diferentes ventajas:

- Un programa escrito en el lenguaje ensamblador requiere considerablemente menos memoria y tiempo de ejecución que un programa escrito en los conocidos lenguajes de alto nivel, como Pascal y C.
- El lenguaje ensamblador da a un programador la capacidad de realizar tareas muy técnicas que serían difíciles, si no es que imposibles de realizar en un lenguaje de alto nivel.
- El conocimiento del lenguaje ensamblador permite una comprensión de la arquitectura de la máquina que ningún lenguaje de alto nivel puede ofrecer.
- Aunque la mayoría de los especialistas en Software desarrolla aplicaciones en lenguajes de alto nivel, que son más fáciles de escribir y de dar mantenimiento, una práctica común es recodificar en lenguaje ensamblador aquellas rutinas que han causado cuellos de botella en el procesamiento.
- Los programas residentes y rutinas de servicio de interrupción casi siempre son desarrollados en el lenguaje ensamblador.

Los lenguajes de alto nivel fueron diseñados para eliminar las particularidades de una computadora específica, mientras que un lenguaje ensamblador está diseñado para una computadora específica, o, de manera más correcta, para una familia específica de microprocesadores.

A continuación se listan los requisitos para aprender el lenguaje ensamblador de la PC:

- Tener acceso a una computadora personal de IBM (cualquier modelo) o una compatible.
- Una copia del sistema operativo MS-DOS o PC-DOS y estar familiarizados con su uso.
- Una copia de un programa ensamblador. Las versiones de Microsoft son conocidas como MASM y QuickAssembler. TASM es de Borland y OPTASM es de System.

Para el aprendizaje de lenguaje ensamblador no es necesario lo siguiente:

- Conocimiento previo de un lenguaje de programación, aunque tenerlo puede ayudarle a comprender algunos conceptos de programación más rápido.
- Conocimiento previo de electrónica o circuitería.

2.1.1. UNIDAD CENTRAL DE PROCESO.

La CPU constituye el cerebro de una computadora digital, pues realiza todas las operaciones aritméticas y lógicas sobre los datos y además controla todos los procesos que se desarrollan en la computadora. Por ejemplo, para que se ejecute una instrucción, ésta debe estar en el interior de la CPU, concretamente en la UC y si hay que realizar cálculos, interviene la UAL. Veamos como funciona cada una de ellas.

- **Unidad de Control.**

Para realizar su tarea la UC necesita conocer, por un lado, la instrucción y, por otro, una serie de informaciones adicionales que deberá tener en cuenta para coordinar, de forma correcta, la ejecución de la instrucción. El resultado de la interpretación de dichas informaciones son una serie de órdenes a los diferentes elementos de la computadora.

La UC no emite todas las órdenes a la vez, sino siguiendo una determinada secuencia. Para ello utiliza un elemento que le va indicando el instante en que debe ejecutar una determinada fase de la instrucción. A este elemento se le denomina Reloj, y se dice que sincroniza las acciones de la UC; cuanto más rápido marque el tiempo, más rápida será la ejecución de la instrucción. Sin embargo, hay un límite, ya que, si marca excesivamente rápido, es posible que no puedan cumplir adecuadamente las órdenes de los diferentes elementos, por lo que se producirán errores.

En la figura 2.2 se esquematiza el conjunto de señales que utiliza la UC y las que genera. Como informaciones adicionales a las instrucciones podemos ver los impulsos de reloj y los indicadores de estado. Los indicadores de estado son una serie de bits que se modifican según resultados de las operaciones anteriores guardando una memoria histórica de los acontecimientos precedentes para que, en función de dichos acontecimientos, pueda la UC tomar decisiones.

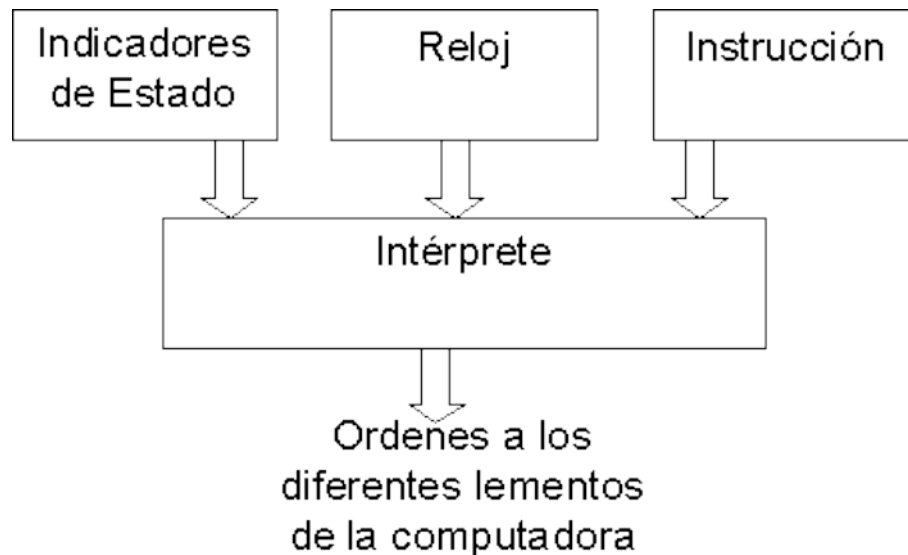


Figura 2.2. Señales que intervienen en la UC.

La unidad de control esta formada, básicamente por un elemento que interpreta las instrucciones y varios elementos de memoria denominados registros. Uno de estos registros almacena la instrucción mientras el intérprete está traduciendo su significado, por lo que se denomina Registro de Instrucción (RI). El resto de las instrucciones permanecen en la memoria, esperando que les toque su turno de ejecución.

La UC por otra parte deberá conocer cuál es la dirección de la próxima instrucción, para poder ir a buscarla una vez que finaliza la ejecución de la instrucción en curso; dirección que guarda el registro llamado Contador de Programa (CP).

Los indicadores de estado están agrupados en un registro denominado Registro de Estado (RE).

La Figura 2.3 muestra los elementos que acabamos de nombrar.

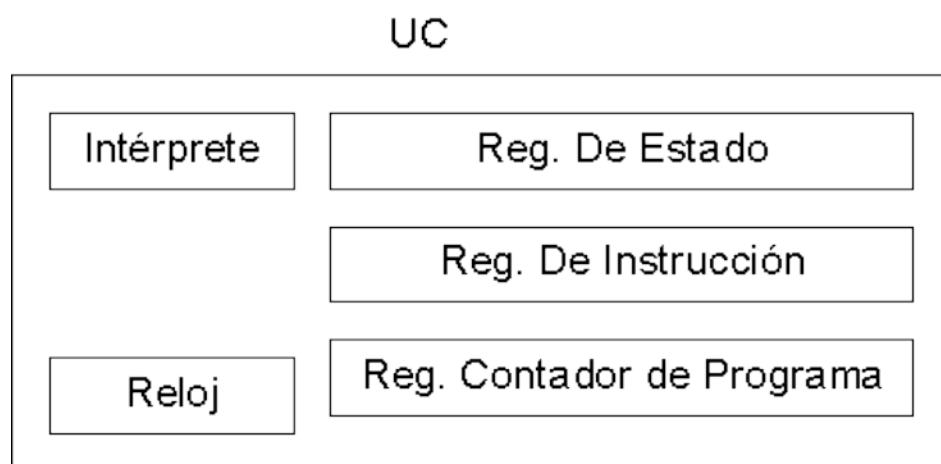


Figura 2.3. Elementos básicos de la Unidad de Control.

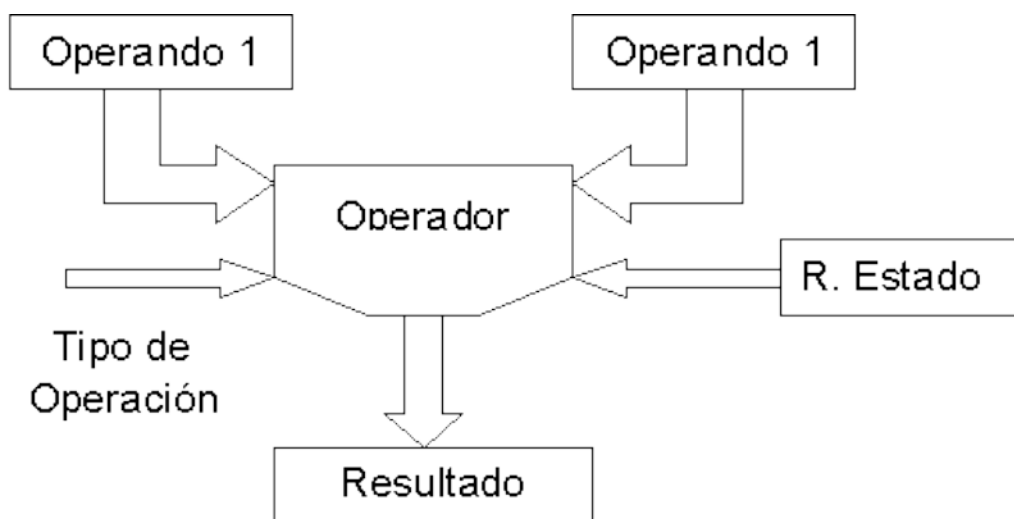
- **Unidad Aritmético - Lógica.**

La unidad Aritmético - Lógica (UAL) es la encargada de realizar los cálculos. Los datos sobre los que se realizan las operaciones se denominan operandos. Al elemento encargado de ejecutar las operaciones se le denomina operador, y esta formado por una serie de circuitos electrónicos que son capaces de sumar dos números binarios o hacer las operaciones lógicas elementales: disyunción, conjunción y negación; incluso algunos operadores son también capaces de multiplicar, dividir y realizar otras operaciones mas complejas.

Para que el operador realice la operación, los operandos se llevan a la UAL y se guardan en unos registros denominados registros de trabajo. El resultado de la operación se guarda también en un registro antes de ser llevado a la memoria o a la Unidad de Entradas y Salidas. Frecuentemente se utiliza un mismo registro para guardar uno de los operandos y, también, el resultado, denominado registro Acumulador.

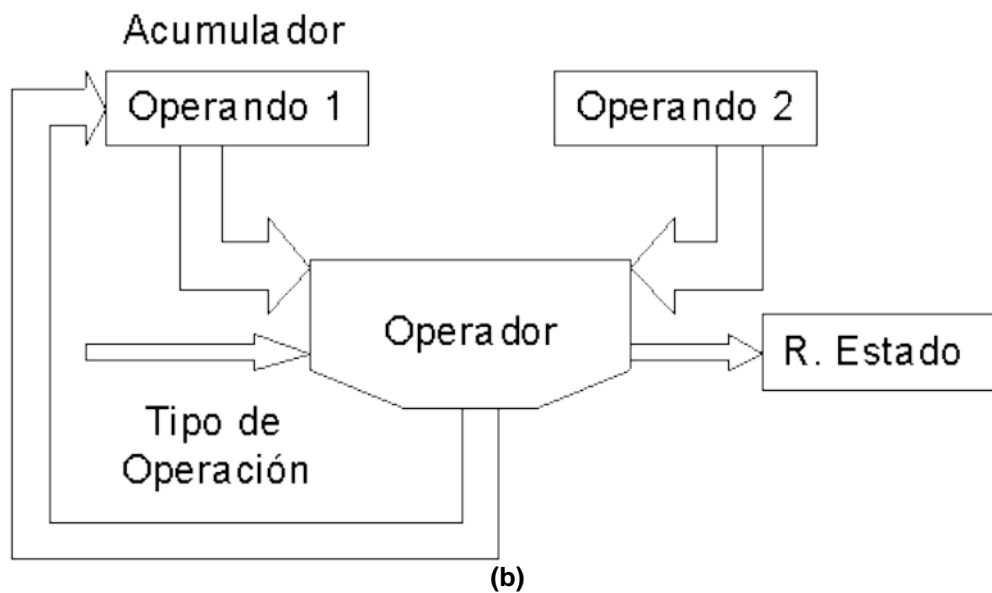
El operador, además de calcular el valor de la operación, modifica el registro de estado según el resultado de la operación. Así, si el resultado es un valor negativo, se modifica un bit de dicho registro, llamado bit negativo o bit N, poniéndose a 1; por el contrario, el bit N permanecerá en estado 0 mientras el contenido del acumulador no sea negativo. De igual forma indicara la UAL a la UC si el resultado ha sido cero, o si ha producido algún acarreo, etc.

En la figura 2.4 se muestran los elementos de la UAL y las señales que intervienen.



(a)

(a) UAL con tres registros: 2 para los operandos y 1 para el resultado.



(b) UAL con acumulador.

Figura 2.4. Elementos y señales de la AUL.

2.1.2. UNIDAD DE MEMORIA PRINCIPAL.

La memoria principal esta formada por un conjunto de unidades llamadas palabras. Dentro de cada una de estas palabras se guarda la información que constituye una instrucción o parte de ella (puede darse el caso de que una sola instrucción necesite varias palabras), o un dato o parte de un dato (también un dato puede ocupar varias palabras).

A la cantidad de palabras que forman la MP se le denomina capacidad de memoria. De este modo, cuanto mayor sea el numero de palabras mayor será el numero de instrucciones y datos que podrá almacenar la computadora.

Una palabra esta formada a su vez de unidades mas elementales llamadas bits, del mismo modo que en el lenguaje natural una palabra esta formada por letras. Cada bit solo puede guardar dos valores, el valor 0 o el valor 1; por eso se dice que son elementos binarios.

El numero de bits que forman una palabra se llama longitud de palabra. Por regla general, las computadoras potentes tienen memorias con longitud de palabra grande, mientras que las computadoras pequeñas tienen memorias con longitud de palabra menor.

En la figura 2.5 se muestra como se puede estar organizada una Memoria Principal.

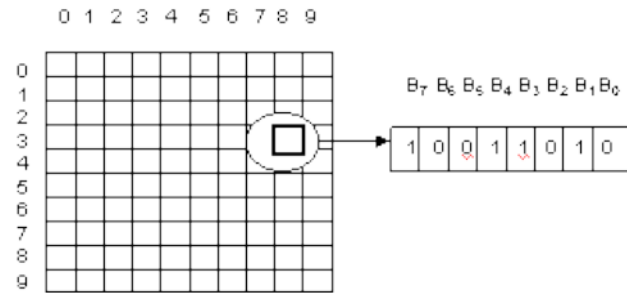


Figura 2.5. Organización de una unidad de memoria.

Las palabras forman una matriz de 10 filas y 10 columnas. La primera palabra corresponderá con la dirección 00, la segunda con la 01, y la última, con la 99. La capacidad de la memoria será de $10 * 10 = 100$ palabras. También se muestra la longitud de la palabra 38, que es de 8 bits, al igual que las demás, y la información que contiene, que es el valor binario 10011010.

Las palabras se distinguen entre sí por la posición que ocupan en la MP, y se puede guardar una información y luego recuperarla indicando el número de dicha posición. A los números que señalan las posiciones de memoria se les da el nombre de direcciones de memoria.

La acción de guardar una información en una palabra de la memoria se llama acceso de escritura, y la acción de recuperarla, acceso de lectura. Los accesos son coordinados por la UC. La secuencia de órdenes que debe generar la UC se indica en la tabla 2.1.

En la tabla 2.1 (a) se muestra un acceso de escritura. Obsérvese que la UC debe indicar, además de la posición donde se debe guardar el dato, el valor del dato y las indicaciones de control que le digan a la memoria que se desea guardar el dato y el momento en que debe iniciarse la operación de escritura.

Esta última orden la dará la UC cuando esté segura de que los datos anteriores han llegado correctamente a la MP. Después de esta última orden, la UC espera un tiempo para asegurar que se ha escrito la información en la MP.

En la tabla 2.1 (b) se muestra cómo se realiza un acceso de lectura. En este caso, la UC no indica el dato, puesto que es precisamente lo que espera recibir. Los demás pasos son idénticos a los del acceso de escritura.

Desde que se inicia la secuencia hasta que finaliza transcurre un tiempo, denominado tiempo de acceso, cuya duración depende de la tecnología con que está fabricada la MP.

Tabla 2.1. Secuencia de acceso a la memoria.

1	La UC indica a la MP que desea escribir.
2	La UC envía la dirección en la que se va a poner el dato.
3	La UC envía el valor del dato.
4	La UC da la orden de inicio del acceso.
5	La UC espera el tiempo necesario para que la MP realice la escritura.

1	La UC indica a la MP que desea leer.
2	La UC envía a la dirección donde está el dato.
3	La UC da la orden de inicio del acceso.
4	La UC espera el tiempo necesario para que la MP realice la lectura.

(a) Acceso de escritura.

(b) Acceso de lectura.

2.1.3 UNIDAD DE ENTRADAS Y SALIDAS.

Ya se ha visto en las secciones precedentes como funcionan la CPU y la MP, pero puede decirse que es necesaria la comunicación entre el interior de la computadora y su entorno o periferia. Esta comunicación se consigue a través de dispositivos de muy diversos tipos, como son: teclados, impresoras, pantallas, discos magnéticos, etc. Es estos dispositivos se les conoce con el nombre genérico de periféricos.

En la figura 2.6 se muestran algunos periféricos conectados a la Unidad de E/S, la cual hace de intermediaria entre los periféricos y la CPU. Las flechas indican el sentido en que fluye la información.

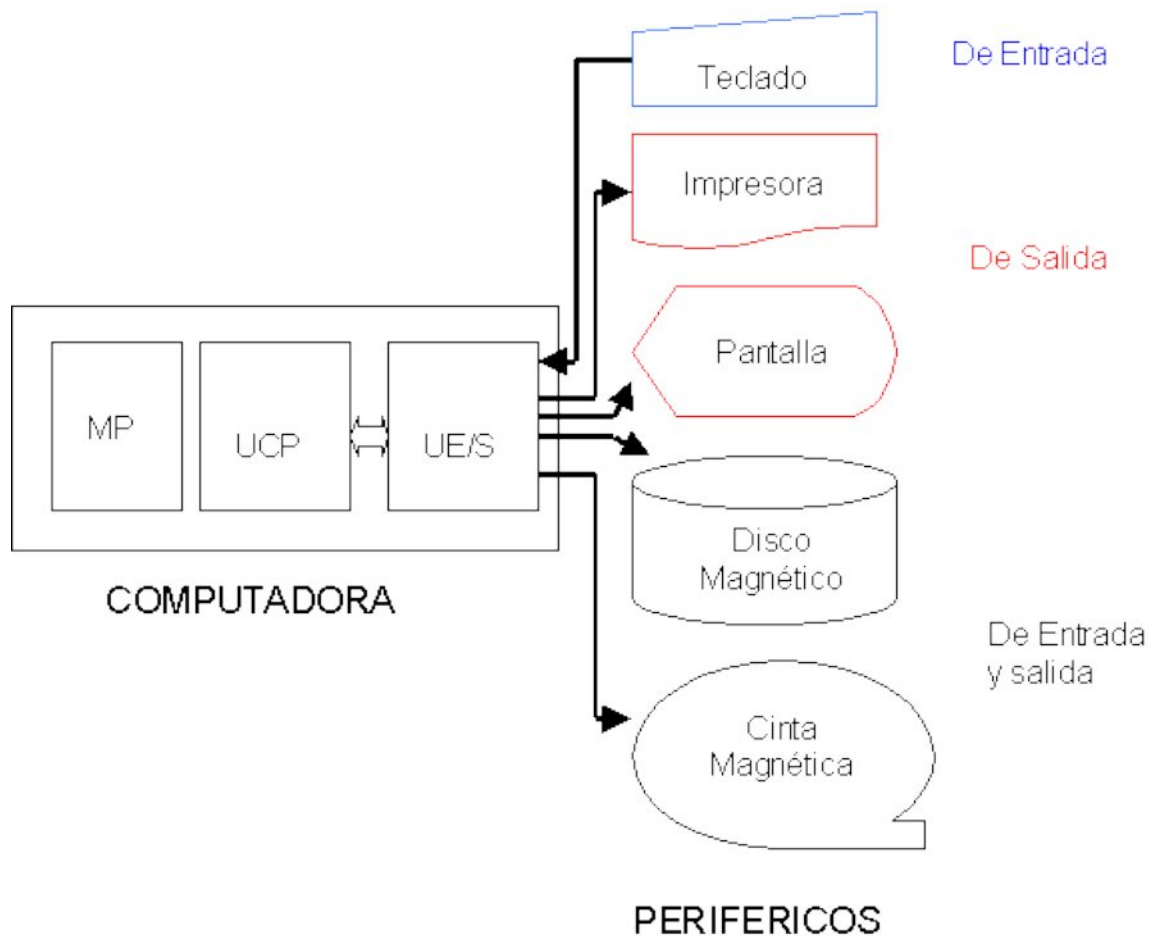


Figura 2.6. La unidad de E/S hace de intermediaria entre la UCP y los periféricos.

La coordinación de la comunicación entre los periféricos y la CPU la realiza la Unidad de E/S. Obsérvese que esta no es un periférico sino un dispositivo que gestiona a los periféricos siguiendo las ordenes de la CPU; es decir, la Unidad de E/S recibe de la Unidad de Control información sobre el tipo de transferencia de datos que debe realizar (si es de entrada o de salida) y periférico que debe de utilizar; si es de salida recibirá también el dato que debe enviar y el momento de la operación.

Entonces, la Unidad de E/S seleccionara el periférico y ejecutara la operación teniendo en cuenta las características propias de cada periférico. Una vez ejecutada la orden avisara a la UC de la terminación de la transferencia.

Cada periférico o parte de un periférico tendrá asignado un numero o dirección que servirá para identificarlo. Cuando la UC quiera seleccionarlo enviara dicho numero a la Unidad de E/S.

Para solucionar el problema de la imposibilidad de saber cuanto tiempo durara una transferencia de información con un periférico se han desarrollado diversas técnicas de comunicación entre la CPU y los periféricos.

Lo mas sencillo es que la CPU, cuando desee hacer una transferencia de información con un periférico, pregunte a la Unidad de E/S si dicho periférico se encuentra disponible. Si no lo esta,

debe repetir la pregunta una y otra vez hasta obtener una respuesta afirmativa, en cuyo caso se inicia la transferencia de información.

Si se desea obtener mayor rendimiento del ordenador, se puede emplear otro método que se denomina sincronización mediante interrupción. La característica de este método es que la CPU, en lugar de dedicarse a preguntar a la Unidad de E/S por el periférico que desea utilizar, lo que hace es indicar a la Unidad de E/S que desea hacer una transferencia con el periférico, y seguidamente, si no está el periférico preparado, empieza otra tarea, olvidándose momentáneamente del periférico.

Cuando este preparado, la Unidad de E/S indicara a la CPU que puede realizarse la transferencia; entonces, la CPU interrumpirá la tarea que esta realizando y atenderá al periférico. De esta forma, la CPU no pierde tiempo esperando al periférico.

2.1.4. INTERCONEXION DE LAS UNIDADES FUNCIONALES.

En los apartados anteriores hemos visto las funciones que realiza cada unidad de la computadora, y que se deben de comunicar entre ellas para pasarse información. Estas informaciones se transmiten en forma de señales eléctricas y, por tanto, circulan a través de conductores eléctricos.

Ya sabemos que en un sistema digital las informaciones se encuentran codificadas en binario formando grupos de bits. Cada bit de uno de estos grupos se suele enviar de una unidad a otra por un conductor diferente al del resto de los bits. De esta forma, si el grupo esta constituido por 8 bits, serán necesarios 8 conductores; si lo forman 16 bits, se emplearan 16 conductores; etc. Se hace así para ahorrar tiempo, ya que circulan todos los bits de un mismo grupo a la vez, cada uno por su conductor. A esta forma de enviar los datos se denomina transmisión en paralelo.

Cuando la CPU quiere leer o escribir en una determinada posición de memoria, enviara la dirección, en paralelo, a la memoria por un conjunto de conductores eléctricos al que se denomina bus de direcciones, ya que se utiliza exclusivamente para enviar direcciones. Así mismo, los datos leídos o los que se quieren escribir se trasladan mediante un conjunto de conductores eléctricos denominado bus de datos. Las ordenes de la CPU a la MP y las respuestas de la MP a la CPU son enviadas por un tercer conjunto de conductores llamado bus de control.

En la figura 2.7 se muestran las conexiones de la CPU con la MP. Se han dibujado en la figura 2.7 (a) todos los conductores eléctricos suponiendo que se emplean 8 hilos para el bus de datos, 16 para el de direcciones y 3 para el de control. En la practica no se dibujan todos los hilos para simplificar las figuras y se hace como en las figuras 2.7 (b) y 2.7 (c).

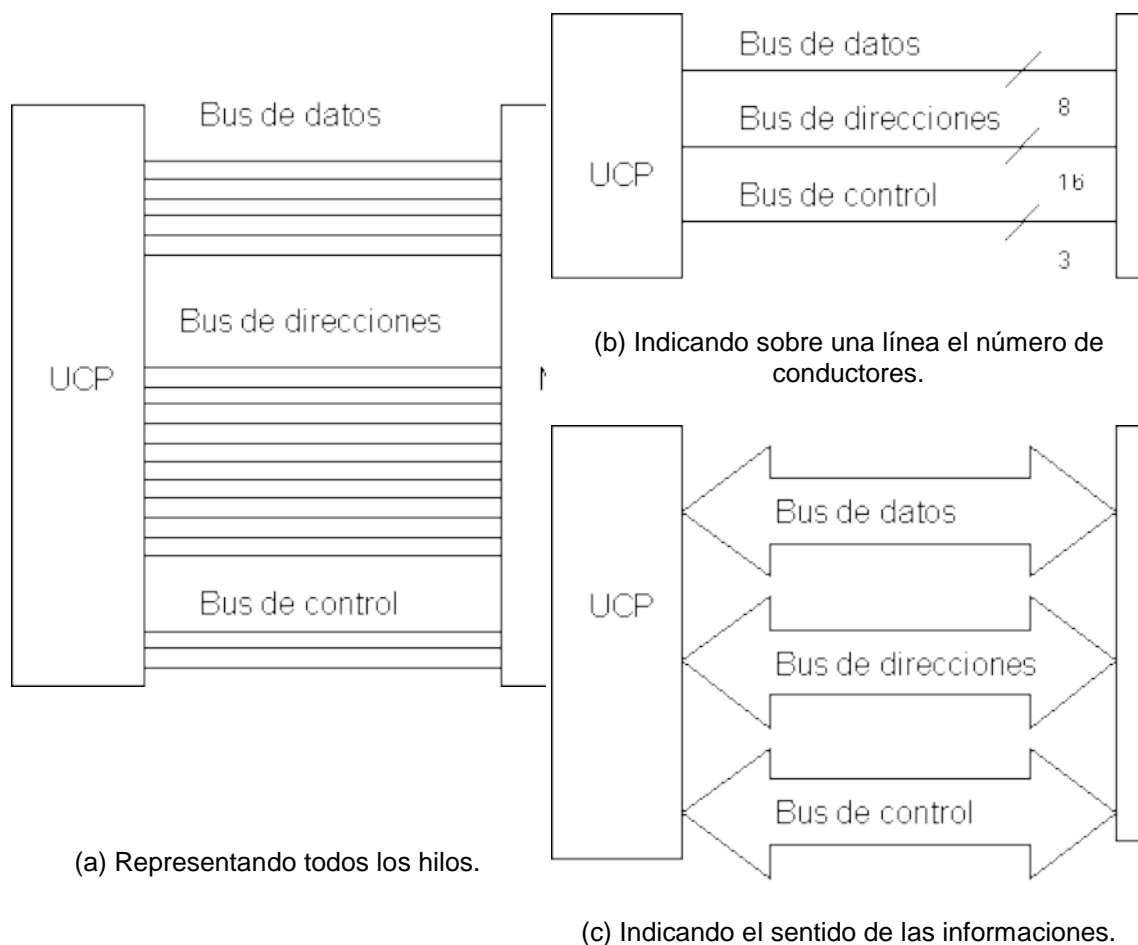
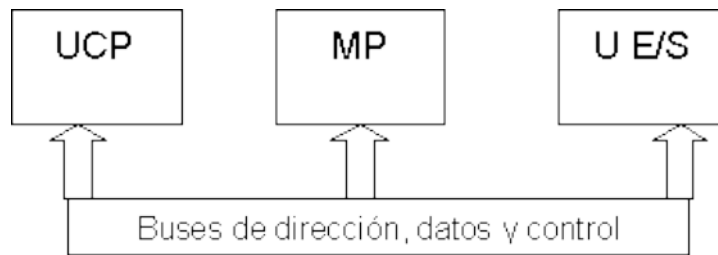


Figura 2.7. Interconexiones entre la UCP y la MP.

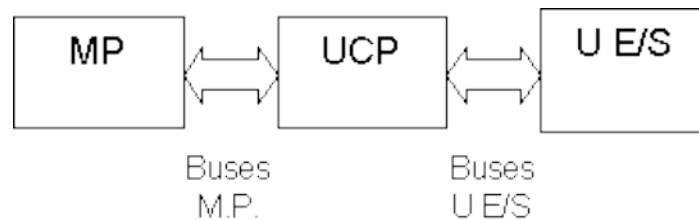
Existen varias formas de interconectar las unidades funcionales de la computadora entre sí. Una de ellas emplea un único conjunto de buses, como puede apreciarse en la figura 2.8 (a), al que se conectan todas las unidades. Esto obliga a que, en un determinado instante, solo dos unidades puedan estar haciendo uso del mismo; si hay que enviar algo a la tercera unidad, se deberá esperar a que quede libre.

La principal ventaja de este sistema de interconexión es su bajo costo, y la principal desventaja, su mayor lentitud. Además, si se emplea el mismo bus de direcciones para indicar la de un periférico y la de una posición de memoria, no podrá coincidir el número de ningún periférico con ninguna dirección de memoria; si así lo hicieran, se producirían errores al seleccionarse a la vez el periférico y la posición de memoria. Por lo tanto, se tendrá cuidado en asignar a los periféricos direcciones que no coincidan con las de la MP. En estos casos se dice que los periféricos están mapeados en memoria, ya que la CPU no distingue entre las transferencias a periféricos y a MP.

Otra forma de interconectar las unidades de la computadora consiste en emplear conjuntos de buses diferentes para la Unidad de E/S y la MP. De esta forma, aunque se encarece el sistema, se gana en velocidad y se pueden emplear direcciones iguales para nombrar periféricos y a posiciones de memoria. La CPU, si quiere seleccionar un periférico, utiliza el bus de la Unidad de E/S, y si quiere seleccionar una posición de memoria, utiliza el bus de memoria. En este caso se dice que los periféricos no están mapeados en memoria. En la figura 2.8 (b) se muestra una posible conexión con el bus múltiple.



(a) Empleando un solo conjunto de buses.



(b) Empleando dos conjuntos de buses diferentes.

Figura 2.8. Formas de interconexión.

2.2. REGISTROS INTERNOS DEL PROCESADOR.

Los registros del procesador se emplean para controlar instrucciones en ejecución, manejar direccionamiento de memoria y proporcionar capacidad aritmética. Los registros son direccionables por medio de un nombre. Los bits por convención, se numeran de derecha a izquierda, como en:

... 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0

Registros de segmento

Un registro de segmento tiene 16 bits de longitud y facilita un área de memoria para direccionamiento conocida como el segmento actual.

Registro CS. El DOS almacena la dirección inicial del segmento de código de un programa en el registro CS. Esta dirección de segmento, mas un valor de desplazamiento en el registro apuntador de instrucción (IP), indica la dirección de una instrucción que es buscada para su ejecución.

Registro DS. La dirección inicial de un segmento de datos de programa es almacenada en el registro DS. En términos sencillos, esta dirección, mas un valor de desplazamiento en una instrucción, genera una referencia a la localidad de un byte específico en el segmento de datos.

Registro SS. El registro SS permite la colocación en memoria de una pila, para almacenamiento temporal de direcciones y datos. El DOS almacena la dirección de inicio del segmento de pila de un programa en el registro SS. Esta dirección de segmento, mas un valor de desplazamiento en el registro del apuntador de pila (SP), indica la palabra actual en la pila que esta siendo direccionada.

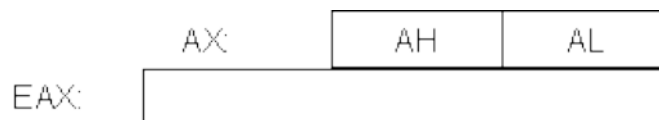
Registros ES. Algunas operaciones con cadenas de caracteres (datos de caracteres) utilizan el registro extra de segmento para manejar el direccionamiento de memoria. En este contexto, el registro ES esta asociado con el registro DI (índice). Un programa que requiere el uso del registro ES puede inicializarlo con una dirección de segmento apropiada.

Registros FS y GS. Son registros extra de segmento en los procesadores 80386 y posteriores.

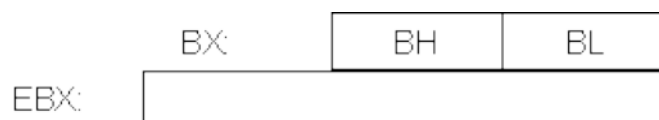
Registros de propósito general.

Los registros de propósito general AX, BX, CX y DX son los caballos de batalla del sistema. Son únicos en el sentido de que se puede direccionarlos como una palabra o como una parte de un byte. El ultimo byte de la izquierda es la parte "alta", y el ultimo byte de la derecha es la parte "baja". Por ejemplo, el registro CX consta de una parte CH (alta) y una parte CL (baja), y usted puede referirse a cualquier parte por su nombre.

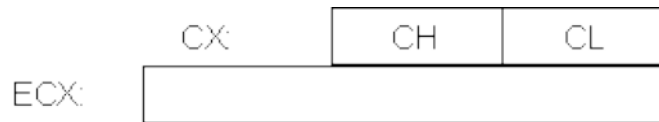
Registro AX. El registro AX, el acumulador principal, es utilizado para operaciones que implican entrada/salida y la mayor parte de la aritmética. Por ejemplo, las instrucciones para multiplicar, dividir y traducir suponen el uso del AX. También, algunas operaciones generan código mas eficiente si se refieren al AX en lugar de a los otros registros.



Registro BX. El BX es conocido como el registro base ya que es el único registro de propósito general que puede ser índice para direccionamiento indexado. También es común emplear el BX para cálculos.



Registro DX. El DX es conocido como el registro de datos. Algunas operaciones de entrada/salida requieren uso, y las operaciones de multiplicación y división con cifras grandes suponen al DX y al AX trabajando juntos.



Puede usar los registros de propósito general para suma y resta de cifras de 8, 16 o 32 bits.

Registro de Apuntador de Instrucciones.

El registro apuntador de instrucciones (IP) de 16 bits contiene el desplazamiento de dirección de la siguiente instrucción que se ejecuta. El IP está asociado con el registro CS en el sentido de que el IP indica la instrucción actual dentro del segmento de código que se está ejecutando actualmente. Los procesadores 80386 y posteriores tienen un IP ampliado de 32 bits, llamado EIP.

En el ejemplo siguiente, el registro CS contiene 25A4[0]H y el IP contiene 412H. Para encontrar la siguiente instrucción que será ejecutada, el procesador combina las direcciones en el CS y el IP:

Segmento de dirección en el registro CS:	25A40H
Desplazamiento de dirección en el registro IP:	+ 412H
Dirección de la siguiente instrucción:	25E52H

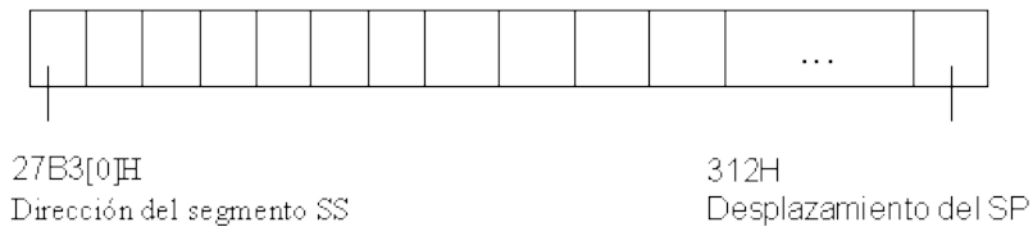
Registros Apuntadores.

Los registros SP (apuntador de la pila) y BP (apuntador de base) están asociados con el registro SS y permiten al sistema acceder datos en el segmento de la pila.

Registro SP. El apuntador de la pila de 16 bits está asociado con el registro SS y proporciona un valor de desplazamiento que se refiere a la palabra actual que está siendo procesada en la pila. Los procesadores 80386 y posteriores tienen un apuntador de pila de 32 bits, el registro ESP. El sistema maneja de forma automática estos registros.

En el ejemplo siguiente, el registro SS contiene la dirección de segmento 27B3[0]H y el SP el desplazamiento 312H. Para encontrar la palabra actual que está siendo procesada en la pila, la computadora combina las direcciones en el SS y el SP:

Dirección de segmento en el registro SS:	27B30H
Desplazamiento en el registro SP:	+312H
Dirección en la pila:	27E42H



Registro BP. El BP de 16 bits facilita la referencia de parámetros, los cuales son datos y direcciones transmitidos vía pila. Los procesadores 80386 y posteriores tienen un BP ampliado de 32 bits llamado el registro EBP.

Registros Índice.

Los registros SI y DI están disponibles para direccionamiento indexado y para sumas y restas.

Registro SI. El registro índice fuente de 16 bits es requerido por algunas operaciones con cadenas (de caracteres). En este contexto, el SI está asociado con el registro DS. Los procesadores 80386 y posteriores permiten el uso de un registro ampliado de 32 bits, el ESI.

Registro DI. El registro índice destino también es requerido por algunas operaciones con cadenas de caracteres. En este contexto, el DI está asociado con el registro ES. Los procesadores 80386 y posteriores permiten el uso de un registro ampliado de 32 bits, el EDI.

Registro de Banderas.

De los 16 bits del registro de banderas, nueve son comunes a toda la familia de procesadores 8086, y sirven para indicar el estado actual de la máquina y el resultado del procesamiento. Muchas instrucciones que piden comparaciones y aritmética cambian el estado de las banderas, algunas cuyas instrucciones pueden realizar pruebas para determinar la acción subsecuente. En resumen, los bits de las banderas comunes son como sigue:

OF (Overflow, desbordamiento). Indica desbordamiento de un bit de orden alto (más a la izquierda) después de una operación aritmética.

DF (dirección). Designa la dirección hacia la izquierda o hacia la derecha para mover o comparar cadenas de caracteres.

IF (interrupción). Indica que una interrupción externa, como la entrada desde el teclado, sea procesada o ignorada.

TF (trampa). Permite la operación del procesador en modo de un paso. Los programas depuradores, como el DEBUG, activan esta bandera de manera que usted pueda avanzar en la ejecución de una sola instrucción a un tiempo, para examinar el efecto de esa instrucción sobre los registros de memoria.

SF (signo). Contiene el signo resultante de una operación aritmética (0 = positivo y 1 = negativo).

ZF (cero). Indica el resultado de una operación aritmética o de comparación (0 = resultado diferente de cero y 1 = resultado igual a cero).

AF (acarreo auxiliar). Contiene un acarreo externo del bit 3 en un dato de 8 bits para aritmética especializada.

PF (paridad). Indica paridad par o impar de una operación en datos de 8 bits de bajo orden (mas a la derecha).

CF (acarreo). Contiene el acarreo de orden mas alto (mas a la izquierda) después de una operación aritmética; también lleva el contenido del ultimo bit en una operación de corrimiento o de rotación.

Las banderas están en el registro de banderas en las siguientes posiciones:

Num. De bit	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Bandera:					O	D	I	T	S	Z		A		P		C

Las banderas mas importantes para la programación en ensamblador son O, S, Z y C, para operaciones de comparación y aritméticas, y D para operaciones de cadenas de caracteres. Los procesadores 80286 y posteriores tienen algunas banderas usadas para propósitos internos, en especial las que afectan al modo protegido. Los procesadores 80286 y posteriores tienen un registro extendido de banderas conocido como Eflags.

3.1. SEGMENTO.

Un segmento es un área especial en un programa que inicia en un limite de un párrafo, esto es, en una localidad de regularmente divisible entre 16, o 10 hexadecimal. Aunque un segmento puede estar ubicado casi en cualquier lugar de la memoria y, en modo real, puede ser hasta de 64K, solo necesita tanto espacio como el programa requiera para su ejecución.

Un segmento en modo real puede ser de hasta 64K. Se puede tener cualquier numero de segmentos; para direccionar un segmento en particular basta cambiar la dirección en el registro del segmento apropiado. Los tres segmentos principales son los segmentos de código, de datos y de la pila.

- **Segmento de código.**

El segmento de código (CS) contiene las instrucciones de maquina que son ejecutadas por lo común la primera instrucción ejecutable esta en el inicio del segmento, y el sistema operativo enlaza a esa localidad para iniciar la ejecución del programa. Como su nombre indica, el

registro del CS direcciona el segmento de código. Si su área de código requiere mas de 64K, su programa puede necesitar definir mas de un segmento de código.

- **Segmento de datos.**

El segmento de datos (DS) contiene datos, constantes y áreas de trabajo definidos por el programa. El registro DS direcciona el segmento de datos. Si su área de datos requiere mas de 64K, su programa puede necesitar definir mas de un segmento de datos.

- **Segmento de pila.**

En términos sencillos, la pila contiene los datos y direcciones que usted necesita guardar temporalmente o para uso de sus "llamadas" subrutinas. El registro de segmento de la pila (SS) direcciona el segmento de la pila.

LIMITES DE LOS SEGMENTOS.

Los registros de segmentos contienen la dirección inicial de cada segmento. La figura 3.1 presenta un esquema de los registros CS, DS y SS; los registros y segmentos no necesariamente están en el orden mostrado. Otros registros de segmentos son el ES (segmento extra) y, en los procesadores 80386 y posteriores, los registros FS y GS, que contienen usos especializados.

Como ya dijimos, un segmento inicia en un limite de párrafo, que es una dirección por lo común divisible entre el 16 decimal o 10 hexadecimal. Suponga que un segmento de datos inicia en la localidad de memoria 045F0H.

Ya que en este y todos los demás casos el ultimo dígito hexadecimal de la derecha es cero, los diseñadores de computadora decidieron que seria innecesario almacenar el dígito cero en el registro del segmento. Así, 045F0H se almacena como 045F, con el cero de la extrema derecha sobrentendido. En donde sea apropiado, el texto indica al cero de la derecha con corchetes, como 045F[0].

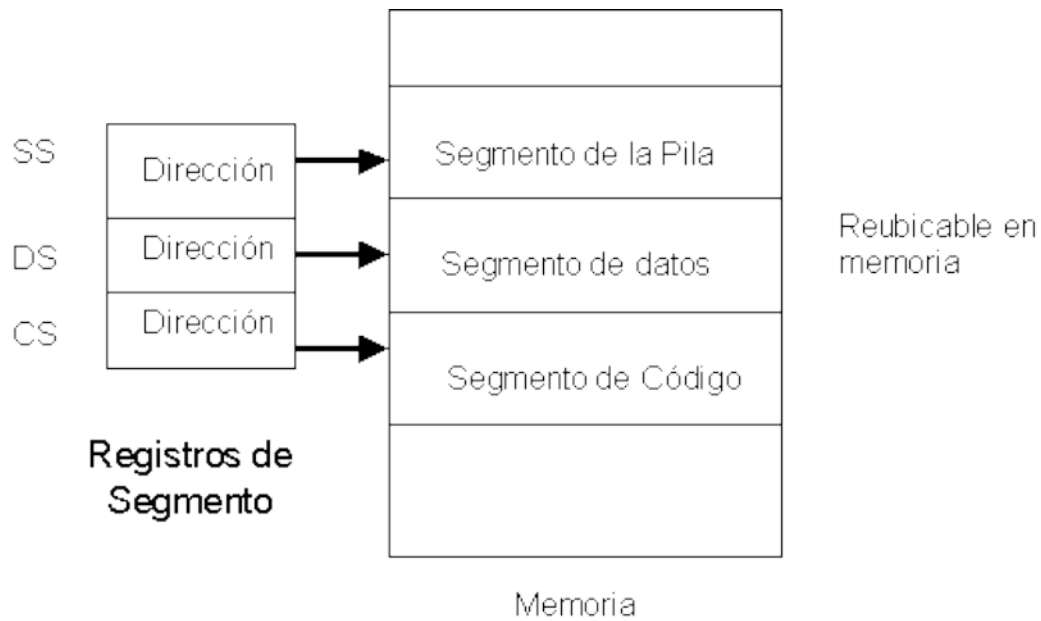


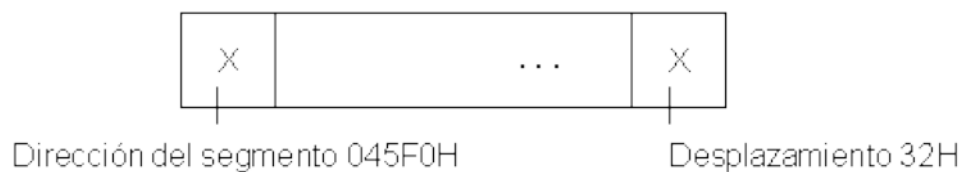
Figura 3.1. Segmentos y registros.

3.2. DESPLAZAMIENTO.

En un programa, todas las localidades de memoria están referidas a una dirección inicial de segmento. La distancia en bytes desde la dirección del segmento se define como el desplazamiento (offset).

Un desplazamiento de dos bytes (16 bits) puede estar en el rango de 0000H hasta FFFFH, o bien, desde cero hasta 65, 535. Así el primer byte del segmento de código tiene un desplazamiento 00, el segundo byte tiene un desplazamiento 01, etc. hasta el desplazamiento 65, 535. Para referir cualquier dirección de memoria en un segmento, el procesador combina la dirección del segmento en un registro de segmento con un valor de desplazamiento.

En el ejemplo siguiente, el registro DS contiene la dirección de segmento del segmento de datos en 045F0H y una instrucción hace referencia a una localidad con un desplazamiento de 0032H bytes dentro del segmento de datos.



Por lo tanto, la localidad real de memoria del byte referido por la instrucción es 04622H;

Dirección del segmento DS: 045F0H
Desplazamiento: +0032H
Dirección real: 04622H

Note que un programa tiene uno o mas segmentos, los cuales pueden iniciar casi en cualquier lugar de memoria, variar en tamaño y estar en cualquier orden.

3.3. METODOS DE DIRECCIONAMIENTO.

El campo de operación de una instrucción especifica la operación que se va a ejecutar. Esta operación debe realizarse sobre algunos datos almacenados en registros de computadora o en palabras de memoria. La manera en que eligen los operandos durante la ejecución del programa depende del modo de direccionamiento de la instrucción. El modo de direccionamiento especifica una regla para interpretar o modificar el campo de dirección de la instrucción antes de que se haga la referencia real al operando. Las computadoras utilizan técnicas de modo de direccionamiento para acomodar una o las dos siguientes consideraciones:

1.- Proporcionar al usuario versatilidad de programación al ofrecer facilidades como apuntadores a memoria, contadores para control de ciclo, indexación de datos y reubicación de datos.

2.- Reducir la cantidad de bits en el campo de direccionamiento de la instrucción.

La disponibilidad de los modos de direccionamiento proporciona al programador con experiencia en lenguaje ensamblador la flexibilidad para escribir programas mas eficientes en relación con la cantidad de instrucciones y el tiempo de ejecución.

Para comprender los diferentes modos de direccionamiento que se presentaran en esta sección, es imperativo entender el ciclo de operación básico de la computadora. La unidad de control de una computadora esta diseñada para recorrer un ciclo de instrucciones que se divide en tres fases principales:

1. Búsqueda de la instrucción de la memoria.
2. Decodificar la instrucción.
3. Ejecutar la instrucción.

Hay un registro en la computadora llamado contador de programa o PC, que lleva un registro de las instrucciones del programa almacenado en la memoria. Pc contiene la dirección de la siguiente instrucción que se va a ejecutar y se incrementa cada vez que se recupera una instrucción de la memoria. La decodificación realizada en el paso 2 determina la operación que se va a ejecutar, el modo de direccionamiento de la instrucción y la posición de los operandos.

Después la computadora ejecuta la instrucción y regresa al paso 1 para hacer la búsqueda de la siguiente instrucción en secuencia.

En algunas computadoras el modo de direccionamiento de la instrucción se especifica con un código binario distinto, como se hace con el código de operación. Otras computadoras utilizan un código binario único que representa la operación y el modo de la instrucción. Pueden definirse instrucciones con diversos modos de direccionamiento y, en ocasiones, se combinan dos o mas modos de direccionamiento en una instrucción.

Aunque la mayoría de los modos de direccionamiento modifican el campo de dirección de la instrucción, hay dos modos que no necesitan el campo de dirección. Son los modos implícito e

inmediato.

MODO IMPLICITO.

En este modo se especifican los operandos en forma implícita en la definición de la instrucción. Por ejemplo, la instrucción "complementar acumulador" es una instrucción de modo implícito porque el operando en el registro de acumulador está implícito en la definición de la instrucción. De hecho todas las instrucciones de referencia a registro que utilizan un acumulador son instrucciones de modo implícito.

Las instrucciones de dirección cero en una computadora organizada con pila son instrucciones de modo implícito porque esta implícito que los operandos están en la parte superior de la pila.

MODO INMEDIATO.

En este modo se especifica el operando en la instrucción misma. En otras palabras, una instrucción de modo inmediato tiene un campo operando, en lugar de un campo de dirección. Un campo de operando contiene el operando real que se va a usar junto con la operación especificada en la instrucción. Las instrucciones de modo inmediato son útiles para inicializar registros en un valor constante.

Se menciona antes que el campo de dirección de una instrucción puede especificar una palabra de memoria o un registro de procesador. Cuando el campo de dirección especifica un registro de procesador se dice que la instrucción está en modo de registro.

MODO DE REGISTRO.

En este modo, los operandos están en registros que residen dentro de la CPU. Se selecciona el registro particular de un campo de registro en la instrucción. Un campo k bits puede especificar cualquiera de 2 a la k registros.

MODO INDIRECTO POR REGISTRO.

En este modo la instrucción especifica un registro en la CPU cuyo contenido proporciona la dirección del operando en la memoria. En otras palabras, el registro seleccionado contiene la dirección del operando en lugar del operando mismo. Antes de utilizar una instrucción de modo indirecto por registro, el programador debe asegurarse de que la dirección de memoria del operando está colocada en el registro del procesador con una instrucción previa. Entonces una referencia al registro es equivalente a especificar una dirección de memoria. La ventaja de una instrucción de modo de registro indirecto es que el campo de dirección de la instrucción utiliza menos bits para seleccionar un registro de los que necesitaría para especificar una dirección de memoria en forma directa.

MODO DE DIRECCIONAMIENTO DIRECTO.

En este modo la dirección efectiva es igual a la parte de dirección de la instrucción. El operando reside en memoria y su dirección la proporciona en forma directa el campo de dirección de la instrucción. En una instrucción de tipo brinco el campo de dirección especifica la dirección de transferencia de control del programa real.

MODO DE DIRECCIONAMIENTO INDIRECTO.

En este modo, el campo de dirección de la instrucción proporciona la dirección en que se almacena la dirección efectiva en la memoria. El control recupera la instrucción de la memoria y utiliza su parte de dirección para acceder la memoria una vez mas con el fin de leer la dirección efectiva.

Unos cuantos modos de direccionamiento requieren que el campo de dirección de la instrucción se sume al contenido de un registro específico en la CPU. En estos modos la dirección efectiva se obtiene del calculo siguiente:

Dirección efectiva = Parte de la instrucción + El contenido de registro CPU.

EL registro de CPU utilizado en el calculo puede ser el contador de programa, un registro de índice o un registro base. En cualquier caso tenemos un modo de direccionamiento diferente que se utiliza para una aplicación distinta.

MODO DE DIRECCIONAMIENTO INDEXADO.

En este modo el contenido de un registro índice se suma a la parte de dirección de la instrucción para obtener la dirección efectiva. El registro índice es un registro CPU especial que contiene un valor índice. Un campo de dirección de la instrucción define la dirección inicial del arreglo de datos en la memoria. Cada operando del arreglo se almacena en la memoria en relación con la dirección inicial.

La distancia entre la dirección inicial y la dirección del operando es el valor del índice almacenado en el registro de índice. Cualquier operando en el arreglo puede accederse con la misma instrucción siempre y cuando el registro índice contenga el valor de índice correcto. El registro índice puede incrementarse para facilitar el acceso a operandos consecutivos. Nótese que si una instrucción de tipo índice no incluye un campo de dirección en su formato, la instrucción se convierte al modo de operación de indirecto por registro.

Algunas computadoras dedican un registro de CPU para que funcione exclusivamente como un registro índice. De manera implícita este registro participa cuando se utiliza una instrucción de modo índice. En las computadoras con muchos registros de procesador, cualquiera de los registros de la CPU pueden contener el numero de índice. En tal caso, el registro debe estar especificado en forma explícita en una campo de registro dentro del formato de instrucción.

MODO DE DIRECCIONAMIENTO DE REGISTRO BASE.

En este modo, el contenido de un registro base se suma a la parte de dirección de la instrucción para obtener la dirección efectiva. Esto es similar al modo de direccionamiento indexado, excepto en que el registro se denomina ahora registro base, en lugar de registro índice. La diferencia entre los dos modos esta en la manera en que se usan mas que en la manera en que se calculan. Se considera que un registro base contiene una dirección base y que el campo de dirección de la instrucción proporciona un desplazamiento en relación con esta dirección base. El modo de direccionamiento de registro base se utiliza en las computadoras para facilitar la localización de los programas en memoria.

4.1. CONCEPTO DE INTERRUPCION.

Una interrupción es una operación que suspende la ejecución de un programa de modo que el sistema pueda realizar una acción especial. La rutina de interrupción ejecuta y por lo regular regresa el control al procedimiento que fue interrumpido, el cual entonces reasume su ejecución.

4.2. TABLA DE SERVICIO DE INTERRUPCION.

Cuando la computadora se enciende, el BIOS y el DOS establecen una tabla de servicios de interrupción en las localidades de memoria 000H-3FFH. La tabla permite el uso de 256 (100H) interrupciones, cada una con un desplazamiento:segmento relativo de cuatro bytes en la forma IP:CS.

El operando de una instrucción de interrupción como INT 05H identifica el tipo de solicitud. Como existen 256 entradas, cada una de cuatro bytes, la tabla ocupa los primeros 1,024 bytes de memoria, desde 000H hasta 3FFH. Cada dirección en la tabla relaciona a una rutina de BIOS o del DOS para un tipo específico de interrupción. Por lo tanto los bytes 0-3 contienen la dirección para la interrupción 0, los bytes 4-7 para la interrupción 1, y así sucesivamente:

INT 00H	INT 01H	INT 02H	INT 03H	INT 04H	INT 05H	INT 06H	...
IP:CS	IP:CS	IP:CS	IP:CS	IP:CS	IP:CS	IP:CS	...
00H	04H	08H	0CH	10H	14H	18H	...

4.3. EVENTOS DE UNA INTERRUPCION.

Una interrupción guarda en la pila el contenido del registro de banderas, el CS, y el IP. por ejemplo, la dirección en la tabla de INT 05H (que imprime la que se encuentra en la pantalla cuando el usuario presiona Ctrl + PrtSC) es 0014H (05H x 4 = 14H). La operación extrae la dirección de cuatro bytes de la posición 0014H y almacena dos bytes en el IP y dos en el CS.

La dirección CS:IP entonces apunta al inicio de la rutina en el área del BIOS, que ahora se ejecuta. La interrupción regresa vía una instrucción IRET (regreso de interrupción), que saca de la pila el IP, CS y las banderas y regresa el control a la instrucción que sigue al INT.

Una interrupción guarda en la pila el contenido del registro de banderas, el CS, y el IP. por ejemplo, la dirección en la tabla de INT 05H (que imprime la que se encuentra en la pantalla cuando el usuario presiona Ctrl + PrtSC) es 0014H (05H x 4 = 14H). La operación extrae la dirección de cuatro bytes de la posición 0014H y almacena dos bytes en el IP y dos en el CS.

La dirección CS:IP entonces apunta al inicio de la rutina en el área del BIOS, que ahora se ejecuta. La interrupción regresa vía una instrucción IRET (regreso de interrupción), que saca de la pila el IP, CS y las banderas y regresa el control a la instrucción que sigue al INT.

4.4. TIPOS DE INTERRUPCIONES.

Las interrupciones se dividen en dos tipos las cuales son: Externas y Internas. Una interrupción externa es provocada por un dispositivo externo al procesador. Las dos líneas que pueden señalar interrupciones externas son la línea de interrupción no enmascarable (NMI) y la línea de petición de interrupción (INTR).

La línea NMI reporta la memoria y errores de paridad de E/S. El procesador siempre actúa sobre esta interrupción, aun si emite un CLI para limpiar la bandera de interrupción en un intento por deshabilitar las interrupciones externas. La línea INTR reporta las peticiones desde los dispositivos externos, en realidad, las interrupciones 05H a la 0FH, para cronometro, el teclado, los puertos seriales, el disco duro, las unidades de disco flexibles y los puertos paralelos.

Una interrupción interna ocurre como resultado de la ejecución de una instrucción INT o una operación de división que cause desbordamiento, ejecución en modo de un paso o una petición para una interrupción externa, tal como E/S de disco. Los programas por lo común utilizan interrupciones internas, que no son enmascarables, para accesar los procedimientos del BIOS y del DOS.

4.5. INTERRUPCION DE BIOS.

El BIOS contiene un extenso conjunto de rutinas de entrada/salida y tablas que indican el estado de los dispositivos del sistema. El dos y los programas usuarios pueden solicitar rutinas del BIOS para la comunicación con los dispositivos conectados al sistema. El método para realizar la interfaz con el BIOS es el de las interrupciones de software. A continuación se listan algunas interrupciones del BIOS.

INT 00H: División entre cero. Llamada por un intento de dividir entre cero. Muestra un mensaje y por lo regular se cae el sistema.

INT 01H: Un solo paso. Usado por DEBUG y otros depuradores para permitir avanzar por paso a través de la ejecución de un programa.

INT 02H: Interrupción no enmascarare. Usada para condiciones graves de hardware, tal como errores de paridad, que siempre están habilitados. Por lo tanto un programa que emite una instrucción CLI (limpiar interrupciones) no afecta estas condiciones.

INT 03H: Punto de ruptura. Usado por depuración de programas para detener la ejecución.

INT 04H: Desbordamiento. Puede ser causado por una operación aritmética, aunque por lo regular no realiza acción alguna.

INT 05H: Imprime pantalla. Hace que el contenido de la pantalla se imprima. Emita la INT 05H para activar la interrupción internamente, y presione las teclas Cltr + PrtSC para activarla externamente. La operación permite interrupciones y guarda la posición del cursor.

INT 08H: Sistema del cronometro. Una interrupción de hardware que actualiza la hora del sistema y (si es necesario) la fecha. Un chip temporizador programable genera una interrupción cada 54.9254 milisegundos, casi 18.2 veces por segundo.

INT 09H: Interrupción del teclado. Provocada por presionar o soltar una tecla en el teclado.

INT 0BH, INT 0CH: Control de dispositivo serial. Controla los puertos COM1 y COM2, respectivamente.

INT 0DH, INT 0FH: Control de dispositivo paralelo. Controla los puertos LPT1 y LPT2, respectivamente.

INT 0EH: Control de disco flexible. Señala actividad de disco flexible, como la terminación de una operación de E/S.

INT 10H: Despliegue en vídeo. Acepta el numero de funciones en el AH para el modo de pantalla, colocación del cursor, recorrido y despliegue.

INT 11H: Determinación del equipo. Determina los dispositivos opcionales en el sistema y regresa el valor en la localidad 40:10H del BIOS al AX. (A la hora de encender el equipo, el sistema ejecuta esta operación y almacena el AX en la localidad 40:10H).

INT 12H: Determinación del tamaño de la memoria. En el AX, regresa el tamaño de la memoria de la tarjeta del sistema, en términos de kilobytes contiguos.

INT 13H: Entrada/salida de disco. Acepta varias funciones en el AH para el estado del disco, sectores leídos, sectores escritos, verificación, formato y obtener diagnostico.

4.6. INTERRUPCION DEL DOS.

Los dos módulos del DOS, IO.SYS y MSDOS.SYS, facilitan el uso del BIOS. Ya que proporcionan muchas de las pruebas adicionales necesarias, las operaciones del DOS por lo general son mas fáciles de usar que sus contrapartes del BIOS y por lo común son independientes de la maquina.

IO.SYS es una interfaz de nivel bajo con el BIOS que facilita la lectura de datos desde la memoria hacia dispositivos externos.

MSDOS.SYS contiene un administrador de archivos y proporciona varios servicios. Por ejemplo, cuando un programa usuario solicita la INT 21H, la operación envía información al MSDOS.SYS por medio del contenido de los registros. Para completar la petición, MSDOS.SYS puede traducir la información a una o mas llamadas a IO.SYS, el cual a su vez llama al BIOS. Las siguientes son las relaciones implícitas:



INTERUPCIONES DEL DOS.

Las interrupciones desde la 20H hasta la 3FH están reservadas para operaciones del DOS. A continuación se mencionan algunas de ellas.

INT 20H: Termina programa. Finaliza la ejecución de un programa .COM, restaura las direcciones para Ctr + Break y errores críticos, limpia los bufer de registros y regresa el control al DOS. Esta función por lo regular seria colocada en el procedimiento principal y al salir de el, CS contendría la dirección del PSP. La terminación preferida es por medio de la función 4CH de la INT 21H.

INT 21H: Petición de función al DOS. La principal operación del DOS necesita una función en el AH.

INT 22H: Dirección de terminación. Copia la dirección de esta interrupción en el PSP del programa (en el desplazamiento 0AH) cuando el DOS carga un programa para ejecución. A la

terminación del programa, el DOS transfiere el control a la dirección de la interrupción. Sus programas no deben de emitir esta interrupción.

INT 23H: Dirección de Ctlr + Break. Diseñada para transferir el control a una rutina del DOS (por medio del PSP desplazamiento 0EH) cuando usted presiona Ctlr + Break o Ctlr + c. La rutina finaliza la ejecución de un programa o de un archivo de procesamiento por lotes. Sus programas no deben de emitir esta interrupción.

INT 24H: Manejador de error crítico. Usada por el dos para transferir el control (por medio del PSP desplazamiento 12H) cuando reconoce un error crítico (a veces una operación de disco o de la impresora). Sus programas no deben de emitir esta interrupción.

INT 25H: Lectura absoluta de disco. Lee el contenido de uno o mas sectores de disco.

INT 26H: Escritura absoluta de disco. Escribe información desde la memoria a uno o mas sectores de disco.

INT 27H: Termina pero permanece residente (reside en memoria). Hace que un programa .COM al salir permanezca residente en memoria.

INT 2FH: Interrupción de multiplexion. Implica la comunicación entre programas, como la comunicación del estado de un spooler de la impresora, la presencia de un controlador de dispositivo o un comando del DOS tal como ASSIGN o APPEND.

INT 33H: Manejador del ratón. Proporciona servicios para el manejo del ratón.

5.1.ELEMENTOS BASICOS.

- **COMENTARIOS EN LENGUAJE ENSAMBLADOR.**

El uso de comentarios a lo largo de un programa puede mejorar su claridad, en especial en lenguaje ensamblador, donde el propósito de un conjunto de instrucciones con frecuencia no es claro. Un comentario empieza con punto y coma (;) y, en donde quiera que lo codifique, el ensamblador supone que todos los caracteres a la derecha de esa línea son comentarios. Un comentario puede contener cualquier carácter imprimible, incluyendo el espacio en blanco. Un comentario puede aparecer solo en una línea o a continuación de una instrucción en la misma línea, como lo muestran los dos ejemplos siguientes:

1. ; Toda esta línea es un comentario.
2. ADD AX, BX ; Comentario en la misma línea que la instrucción.

Ya que un comentario aparece solo en un listado de un programa fuente en ensamblador y no genera código de maquina, puede incluir cualquier cantidad de comentarios sin afectar el tamaño o la ejecución del programa ensamblado.

Otra manera de proporcionar comentarios es por medio de la directiva COMMENT.

- **PALABRAS RESERVADAS.**

Ciertas palabras en lenguaje ensamblador están reservadas para sus propósitos propios, y son usadas solo bajo condiciones especiales. Por categorías, las palabras reservadas incluyen:

- Instrucciones, como MOV y ADD, que son operaciones que la computadora puede ejecutar.
- Directivas como END o SEGMENT, que se emplean para proporcionar comandos al ensamblador.
- Operadores, como FAR y SIZE, que se utilizan en expresiones.
- Símbolos predefinidos, como @Data y @Model, que regresan información a su programa.

El uso de una palabra reservada para un propósito equivocado provoca que el ensamblador genere un mensaje de error.
Ver palabras reservadas.

• IDENTIFICADORES.

Un identificador es un nombre que se aplica a elementos en el programa. Los dos tipos de identificadores son: nombre, que se refiere a la dirección de un elemento de dato. y etiqueta, que se refiere a la dirección de una instrucción. Las mismas reglas se aplican tanto para los nombres como para las etiquetas. Un identificador puede usar los siguientes caracteres:

- | | |
|---------------------------|---|
| 1.- Letras del alfabeto: | Desde la A hasta la Z |
| 2.- Dígitos: | Desde el 0 al 9 (no puede ser el primer carácter) |
| 3.- Caracteres especiales | Signo de interrogación (?) |
| | Subrayado (_) |
| | Signo de pesos (\$) |
| | Arroba (@) |
| | Punto (.) (no puede ser el primer carácter) |

El primer carácter de un identificador debe ser una letra o un carácter especial, excepto el punto. Ya que el ensamblador utiliza algunos símbolos especiales en palabras que inician con el símbolo @, debe evitar usarlo en sus definiciones.

El ensamblador trata las letras mayúsculas y minúsculas como iguales. La longitud máxima de un identificador es de 31 caracteres (247 desde el MASM 6.0). Ejemplos de nombres validos son COUNT, PAGE25 y \$E10. Se recomienda que los nombres sean descriptivos y con significado. Los nombres de registros, como AX, DI y AL, están reservados para hacer referencia a esos mismos registros. En consecuencia, en una instrucción tal como:

ADD AX, BX

el ensamblador sabe de forma automática que AX y BX se refieren a los registros. Sin embargo, en una instrucción como:

MOV REGSAVE, AX

el ensamblador puede reconocer el nombre REGSAVE solo si se define en algún lugar del programa.

- **INSTRUCCIONES.**

Un programa en lenguaje ensamblador consiste en un conjunto de enunciados. Los dos tipos de enunciados son:

1. Instrucciones, tal como MOV y ADD, que el ensamblador traduce a código objeto.
2. Directivas, que indican al ensamblador que realiza una acción específica, como definir un elemento de dato.

A continuación está el formato general de un enunciado, en donde los corchetes indican una entrada opcional:

[identificador] operación [operando(s)] [;comentarios]

Un identificador (si existe), una operación y un operando (si existe) están separados por al menos un espacio en blanco o un carácter tabulador. Existe un máximo de 132 caracteres en una línea (512 desde el MASM 6.0), aunque la mayoría de los programadores prefiere permanecer en los 80 caracteres ya que es el número máximo que cabe en la pantalla. A continuación se presentan dos ejemplos de enunciados:

IDENTIFICADOR	OPERACION	OPERANDO	COMENTARIO
Directiva: COUNT	DB	1	;Nom, Op, Operando
Instrucción:	MOV	AX, 0	;Operación, 2 Operand

Identificador, operación y operando pueden empezar en cualquier columna. Sin embargo, si de manera consistente se inicia en la misma columna para estas tres entradas se hace un programa mas legible.

- **IDENTIFICADOR**

Como ya se explico, el termino nombre se aplica al nombre de un elemento o directiva definida, mientras que el termino etiqueta se aplica al nombre de una instrucción.

- **OPERACION**

La operación, que debe ser codificada, es con mayor frecuencia usada para la definición de áreas de datos y codificación de instrucciones. Para un elemento de datos, una operación como DB o DW define un campo, área de trabajo o constante.

- **OPERANDO**

El operando (si existe) proporciona información para la operación que actúa sobre él. Para un elemento de datos, el operando identifica su valor inicial. Por ejemplo, en la definición siguiente de un elemento de datos llamado COUNTER, la operación DB significa "definir byte", y el operando inicializa su contenido con un valor cero:

NOMBRE	OPERACION	OPERANDO	COMENTARIO
COUNTER	DB	0	;Define un byte con el valor 0

Para una instrucción, un operando indica en donde realizar la acción. Un operando de una instrucción puede tener una, dos o tal vez ninguna entrada. Aquí están tres ejemplos:

OPERACION	OPERANDO	COMENTARIO	OPERANDO
RET		;Regresa	Ninguno
INC	CX	;Incrementa CX	Uno
ADD	AX, 12	;Suma 12 al AX	Dos

DIRECTIVAS PARA LISTAR: PAGE Y TITLE

La directiva PAGE y TITLE ayudan a controlar el formato de un listado de un programa en ensamblador. Este es su único fin, y no tienen efecto sobre la ejecución subsecuente del programa.

PAGE. Al inicio de un programa, la directiva PAGE designa el número máximo de líneas para listar en una página y el número máximo de caracteres en una línea. Su formato general es:

PAGE [longitud][, ancho]

El ejemplo siguiente proporciona 60 líneas por página y 132 caracteres por línea:

PAGE 60, 132

El número de líneas por página puede variar desde 10 hasta 255, mientras que el número de caracteres por línea desde 60 hasta 132. La omisión de PAGE causa que el ensamblador tome PAGE 50, 80.

TITLE. Se puede emplear la directiva TITLE para hacer que un título para un programa se imprima en la línea 2 de cada página en el listado del programa. Puede codificar TITLE de una vez, al inicio del programa. Su formato general es:

TITLE Texto.

Para el operando texto, una técnica recomendada es utilizar el nombre del programa como se registra en el disco. Por ejemplo:

TITLE Prog1 Mi primer programa en ensamblador
DIRECTIVA SEGMENT

Un programa ensamblado en formato .EXE consiste en uno o más segmentos. Un segmento de pila define el almacén de la pila, un segmento de datos define los elementos de datos y un segmento de código proporciona un código ejecutable. Las directivas para definir un segmento, SEGMENT y ENDS tienen el formato siguiente:

NOMBRE	OPERACION	OPERANDO	COMENTARIO
Nombre	SEGMENT	[Opciones]	;inicia el segmento
.			
.			
.			
Nombre	ENDS		;Fin del segmento

El enunciado SEGMENT define el inicio de un segmento. El nombre del segmento debe estar presente, ser único y cumplir las convenciones para nombres del lenguaje. El enunciado ENDS indica el final del segmento y contiene el mismo nombre del enunciado SEGMENT. El tamaño máximo de un segmento es de 64K. El operando de un enunciado SEGMENT puede tener tres tipos de opciones: alineación, combinar y clase, codificadas en este formato:

nombre SEGMENT alineación combinar ' clase '

TIPO ALINEACION. La entrada alineación indica el límite en el que inicia el segmento. Para el requerimiento típico, PARA, alinea el segmento con el límite de un párrafo, de manera que la dirección inicial es divisible entre 16, o 10H. En ausencia de un operando hace que el ensamblador por omisión tome PARA.

TIPO COMBINAR. La entrada combinar indica si se combina el segmento con otros segmentos cuando son enlazados después de ensamblar. Los tipos de combinar son STACK, COMMON, PUBLIC y la expresión AT. Por ejemplo, el segmento de la pila por lo común es definido como:

nombre SEGMENT PARA STACK

Puede utilizar PUBLIC y COMMON en donde tenga el propósito de combinar de forma separada programas ensamblados cuando los enlaza. En otros casos, donde un programa no es combinado con otros, puede omitir la opción o codificar NONE.

TIPO CLASE. La entrada clase, encerrada entre apóstrofes, es utilizada para agrupar segmentos cuando se enlazan. Se utiliza la clase 'code' para el segmento de códigos, 'data' por segmento de datos y 'stack' para el segmento de la pila. El ejemplo siguiente define un segmento de pila con tipos alineación, combinar y clase:

```
nombre    SEGMENT    PARA    STACK    'Stack'
```

DIRECTIVA ASSUME.

Un programa utiliza el registro SS para direccionar la pila, al registro DS para direccionar el segmento de datos y el registro CS para direccionar el segmento de código. Para este fin, usted tiene que indicar al ensamblador el propósito de cada segmento en el programa. La directiva para este propósito es ASSEME, codificada en el segmento de código como sigue:

OPERACION	OPERANDO
ASSUME	SS:nompila, DS:nomsegdatos, CS: nomsegcodigo,..

Los operandos pueden aparecer en cualquier orden. Al igual que otras directivas, ASSUME es solo un mensaje que ayuda al ensamblador a convertir código simbólico a código maquina; aun puede tener que codificar instrucciones que físicamente cargan direcciones en registros de segmentos en el momento de la ejecución.

```

1          PAGE 60,132
2          TITLE P04ASM1 ESTRUCTURA DE UN PROGRAMA .EXE
3;-----
4  STACKSG SEGMENT    PARA STACK 'Stack'
5          ...
6  STACKSG ENDS
7;-----
8  DATASG  SEGMENT    PARA 'Data'
9          ...
10 DATASG  ENDS
11;-----
12 CODESG  SEGMENT    PARA 'Code'
13 BEGIN   PROC        FAR
14         ASSUME      SS:STACKSG, DS:DATASG,CS:CODESG
15         MOV         AX, DATASG      ;Obtiene la dirección del segmento de datos
16         MOV         DS, AX          ;Almacena dirección en DS
17         ...
18         MOV         AX, 4C00H       ;Petición
19         INT         21H             ;Salida al DOS
20 BEGIN   ENDP
21 CODESG  ENDS
22         END          BEGIN

```

5.2. DIRECTIVAS SIMPLIFICADAS DE SEGMENTOS

Los ensambladores de MicroSoft y de Borland proporcionan algunas formas abreviadas para definir segmentos. Para usar estas abreviaturas, inicialice el modelo de memoria antes de definir algún segmento. El formato general (incluyendo el punto inicial) es:

.MODEL modelo de memoria

El modelo de memoria puede ser TINY, SMALL, MEDIUM, COMPACT o LARGE. Los requisitos para cada modelo son:

MODELO	NUMERO DE SEGMENTOS DE CODIGO	NUMERO DE SEGMENTOS DE DATOS
TINY	*	*
SMALL	1	1
MEDIUM	MAS DE 1	1
COMPACT	1	MAS DE 1
LARGE	MAS DE 1	MAS DE 1

Puede utilizar cualquiera de estos modelos para un programa autónomo (esto es, un programa que no este enlazado con algún otro). El modelo TINY esta destinado para uso exclusivo de programas .COM, los cuales tienen sus datos, código y pila en un segmento. El modelo SMALL exige que el código quepa en un segmento de 64K y los datos en otro segmento de 64K. La directiva .MODEL genera automáticamente el enunciado ASSUME necesario.

Los formatos generales (incluyendo el punto inicial) para las directivas que define los segmentos de la pila, de datos y de código son:

.STACK [tamaño]

.DATA

.CODE [nombre]

Cada una de estas directivas hace que el ensamblador genere el enunciado SEGMENT necesario y su correspondiente ENDS. Los nombres por omisión de los segmentos (que usted no tiene que definir) son STACK, DATA y TEXT (para el segmento de código).

La figura 4.3 proporciona un ejemplo haciendo uso de las directivas simplificadas de segmento.

```
page 60,132
TITLE  P04ASM2 (EXE)  Operaciones de mover y sumar
;-----
.MODEL  SMALL
.STACK  64 ;Se define la pila
.DATA   ;Se definen los datos

FLDA   DW      250
FLDB   DW      125
FLDC   DW      ?
;-----

.CODE
BEGIN  PROC    FAR      ;Se define el segmento de código
```


	MOV	AX, @data	;Se asigna la dirección de DATASG (Prog. anterior)
	MOV	AX, FLDA	;Mover 0250 a AX
	ADD	AX, FLDB	;Sumar 0125 a AX
	MOV	FLDC, AX	;Almacenar suma en FLDC
	MOV	AX, 4C00H	;Salida a DOS
	INT	21H	
BEGIN	ENDP		;Fin de procedimiento
	END	BEGIN	;Fin de programa

6.1. TRANSFERENCIA DE DATOS.

La instrucción de transferencia de datos por excelencia es:

MOV destino, fuente

entendiendo por fuente el contenido que se va a transferir a una determinada zona o registro de memoria denominada destino.

Esta instrucción, por tanto, nos va a permitir transferir información entre registros y memoria, memoria y registros y entre los propios registros utilizando alguno de los diferentes modos de direccionamiento. Con la instrucción MOV diremos que se pueden realizar todo tipo de movimientos teniendo en cuenta las siguientes restricciones:

1.- No se puede realizar una transferencia de datos entre dos posiciones de memoria directamente, por esta razón, siempre que queramos efectuarlas tendremos que utilizar un registro intermedio que haga de puente.

Por ejemplo, para hacer la operación $DATO1 \leftarrow DATO2$, la instrucción `MOV DATO2,DATO1` sería incorrecta. Lo que sí sería correcto sería utilizar el registro DX, u otro, como puente y hacer:

```
MOV DX,DATO1
MOV DATO2,DX
```

2.- Tampoco se puede hacer una transferencia directa entre dos registros de segmento. Por eso, como en el caso anterior, si fuera preciso se utilizaría un registro como puente.

3.- Asimismo, tampoco se puede cargar en los registros de segmento un dato utilizando direccionamiento inmediato, es decir, una constante, por lo que también habrá que recurrir a un registro puente cuando sea preciso.

Una instrucción útil pero no imprescindible es:

XCHG DATO1, DATO2

que intercambia los contenidos de las posiciones de memoria o registros representados por DATO1 y DATO2.

Por ejemplo, si queremos intercambiar los contenidos de los registros AX y BX, podemos hacer:

```
MOV AUX, AX
MOV AX, BX
MOV BX, AUX
```

en donde AUX es una variable auxiliar que hace de puente, o simplemente utilizar:

```
XCHG AX, BX
```

Las restricciones que presenta esta operación es que no se pueden efectuar intercambios directamente entre posiciones de memoria ni tampoco entre registros de segmento.

La instrucción XLAT carga en el registro AL el contenido de la posición [BX][AL], en donde el registro BX ha de apuntar al comienzo de una tabla. Dichio de otra manera, AL hace de índice de la tabla y de almacén destino del contenido de la tabla.

Por ejemplo, el siguiente programa:

```
DATOS SEGMENT
    TABLA DB 2,3,5,8,16,23
DATOS ENDS
CODIGO SEGMENT
    MOVE BX, OFFSET TABLA ; Inicializa BX con la dirección donde comienza la tabla
    MOVE AL, 5
    XLAT TABLA
CODIGO ENDS
```

hace que al final el contenido de AL se 16 ya que es el 5to. elemento de la tabla y AL antes de XLAT TABLA contenía el valor 5.

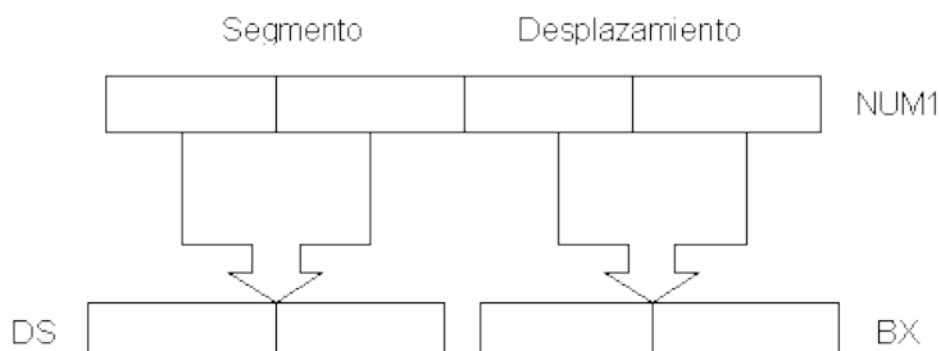
Para finalizar con las instrucciones de transferencia veremos un grupo de tres instrucciones:

- LEA o cargar dirección efectiva.
- LDS o cargar el puntero en DS.
- LES o cargar el puntero en ES.

denominadas de transferencia de direcciones.

La primera, LEA, carga el desplazamiento u OFFSET correspondiente al operando fuente en el operando destino. Por ejemplo, la instrucción MOVE BX, OFFSET TABLA del ejemplo anterior sería equivalente a LEA BX, TABLA.

La segunda, LDS, se utiliza para cargar el valor del segmento de una variable en el registro DS y el desplazamiento correspondiente en el registro o posición de memoria indicada en la instrucción. Por ejemplo, la instrucción LDS BX, NUM1 haría esquemáticamente lo siguiente:



La tercera y ultima de las instrucciones, LES, es similar a LDS, con la única salvedad de que el valor del segmento se carga sobre el registro de segmento ES en vez del DS.

6.2. SUMA Y RESTA.

Las instrucciones ADD y SUB realizan sumas y restas sencillas de datos binarios. Los números binarios negativos están representados en la forma de complemento a dos: Invierta todos los bits del numero positivo y sume 1. Los formatos generales para las instrucciones ADD y SUB son:

[etiqueta]	ADD/SUB	{registro, registro}
[etiqueta]	ADD/SUB	{memoria, registro}
[etiqueta]	ADD/SUB	{registro, memoria}
[etiqueta]	ADD/SUB	{registro, inmediato}
[etiqueta]	ADD/SUB	{memoria, inmediato}

Como en otras instrucciones, no existen operaciones directas de memoria a memoria. El ejemplo siguiente utiliza el registro AX para sumar WORDA a WORDB:

```
WORDA DW 123    ;Define WORDA
WORDB DW 25     ;Define WORDB
...
MOV AX, WORDA   ;Mueve WORDA al AX
ADD AX, WORDB   ;Suma WORDB al AX
MOV WORDB, AX   ;Mueve AX a WORDB
```

La figura 6.1. proporciona ejemplos de ADD y SUB para el procesamiento de valores en un byte y en una palabra. El procedimiento B10ADD utiliza ADD para procesar bytes y el procedimiento C10SUB utiliza SUB para procesar palabras.

```
TITLE      P13ADD (COM) Operaciones ADD y SUB
.MODEL SMALL
.CODE
ORG 100H
BEGIN:     JMP SHORT MAIN
;-----
BYTEA     DB      64H           ;DATOS
BYTEB     DB      40H
BYTEC     DB      16H
WORDA     DW      4000H
WORDB     DW      2000H
WORDC     DW      1000H
;-----
MAIN      PROC  NEAR           ;Procedimiento principal:
          CALL  B10ADD         ;Llama a la rutina ADD
          CALL  C10SUB         ;Llama a la rutina SUB
          INT   21H
MAIN      ENDP
;
; Ejemplos de suma (ADD) de bytes:
;-----
B10ADD    PROC
```

```

        MOV     AL, BYTEA
        MOV     BL, BYTEB
        ADD     AL, BL           ;registro a registro
        ADD     AL, BYTEC       ;memoria a registro
        ADD     BYTEA, BL       ;registro a memoria
        ADD     BL, 10H         ;inmediato a registro
        ADD     BYTEA, 25H      ;inmediato a memoria
        RET
B10ADD ENDP
;      Ejemplos de resta (SUB) de palabras:
;-----
C10SUB PROC
        MOV     AX, WORDA
        MOV     BX, WORDB
        SUB     AX, BX          ;Registro a registro
        SUB     AX, WORDC       ;Memoria de registro
        SUB     WORDA, BX       ;Registro de memoria
        SUB     BX, 1000H       ;Inmediato de registro
        SUB     WORDA, 256H     ;Inmediato de memoria
        RET
C10SUB ENDP
        END BEGIN

```

Desbordamientos

Este alerta con los desbordamientos en las operaciones aritméticas. Ya que un byte solo permite el uso de un bit de signo y siete de datos (desde -128 hasta +127), una operación aritmética puede exceder con facilidad la capacidad de un registro de un byte. Y una suma en el registro AL, que exceda su capacidad puede provocar resultados inesperados.

6.3. OPERANDOS LOGICOS.

La lógica booleana es importante en el diseño de circuitos y tiene un paralelo en la lógica de programación. Las instrucciones para lógica booleana son AND, OR, XOR, TEST y NOT, que pueden usarse para poner bits en 0 o en 1 y para manejar datos ASCII con propósitos aritméticos. El formato general para las operaciones booleanas es:

[etiqueta :] operación {registro/memoria}, {registro/memoria/inmediato}

El primer operando se refiere a un byte o palabra en un registro o memoria y es el único valor que es cambiado. El segundo operando hace referencia a un registro o a un valor inmediato. La operación compara los bits de los dos operandos referenciados y de acuerdo con esto establece las banderas CF, OF, PF, SF y ZF.

AND. Si ambos bits comparados son 1, establece el resultado en 1. Las demás condiciones dan como resultado 0.

OR. Si cualquiera (o ambos) de los bits comparados es 1, el resultado es 1. Si ambos bits están en 0, el resultado es 0.

XOR. Si uno de los bits comparados es 0 y el otro 1, el resultado es 1. Si ambos bits comparados son iguales (ambos 0 o ambos 1), el resultado es 0.

TEST. Establece las banderas igual que lo hace AND, pero no cambia los bits de los operandos.

Las operaciones siguientes AND, OR y XOR ilustran los mismos valores de bits como operandos:

	AND	OR	XOR
	0101	0101	0101
	<u>0011</u>	<u>0011</u>	<u>0011</u>
Resultado:	0001	0111	0110

Es útil recordar la siguiente regla: el empleo de AND con bits 0 es 0 y el de OR con bits 1 es 1.

Ejemplos de operaciones booleanas.

Para los siguientes ejemplos independientes, suponga que AL contiene 11000101 y el BH contiene 01011100:

- 1.- AND AL,BH ;Establece AL a 0100 0100
- 2.- AND AL,00H ;Establece AL a 0000 0000
- 3.- AND AL,0FH ;Establece AL a 0000 0101
- 4.- OR BH,AL ;Establece BH a 1101 1101
- 5.- OR CL,CL ;Pone en uno SF y ZF
- 6.- XOR AL,AL ;Establece AL a 0000 0000
- 7.- XOR AL,0FFH ;Establece AL a 0011 1010

Los ejemplos 2 y 6 muestran formas de limpiar un registro, y ponerlo a cero. El ejemplo 3 pone a cero los cuatro bits mas a la izquierda de AL.

TEST actúa igual que AND, pero solo establece las banderas. Aquí están algunos ejemplos :

- 1.- TEST BL, 11110000B ; Alguno de los bits de mas a la
JNZ ... ; izquierda es BL en diferentes de cero?
- 2.- TEST AL, 00000001B ; AL contiene
JNZ ... ; un numero impar?
- 3.- TEST DX, 0FFH ; El DX contiene
JNZ ... ; un valor cero?

La instrucción NOT.

La instrucción NOT solo invierte los bits en un byte o palabra en un registro o en memoria; esto es, convierte los ceros en unos y los unos en ceros. El formato general es:

[etiqueta:] NOT {registro/memoria}
--

Por ejemplo si el AL contiene 11000101, la instrucción NOT AL cambia el AL a 00111010 (el resultado es el mismo de XOR AL, 0FFH). Las banderas no son afectadas.

6.4. CORRIMIENTO Y ROTACION.

- **CORRIMIENTO DE BITS.**

Las instrucciones de corrimiento, que son parte de la capacidad lógica de la computadora, pueden realizar las siguientes acciones:

1. Hacer referencia a un registro o dirección de memoria.
2. Recorre bits a la izquierda o a la derecha.
3. Recorre hasta 8 bits en un byte, 16 bits en una palabra y 32 bits en una palabra doble.
4. Corrimiento lógico (sin signo) o aritmético (con signo).

El segundo operando contiene el valor del corrimiento, que es una constante (un valor inmediato) o una referencia al registro CL. Para los procesadores 8088/8086, la constante inmediata solo puede ser 1; un valor de corrimiento mayor que 1 debe estar contenido en el registro CL. Procesadores posteriores permiten constantes de corrimiento inmediato hasta 31.

El formato general para el corrimiento es

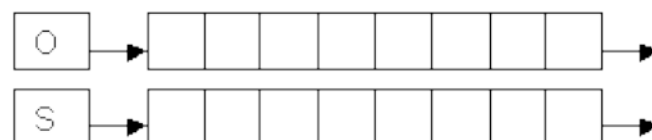
[etiqueta:] Corrim. {registro/memoria}, {CL/inmediato}
--

Corrimiento de bits hacia la derecha.

Los corrimientos hacia la derecha (SHR y SAR) mueven los bits hacia la derecha en el registro designado. El bit recorrido fuera del registro mete la bandera de acarreo. Las instrucciones de corrimiento a la derecha estipulan datos lógicos (sin signo) o aritméticos (con signo):

SHR: Desplazamiento lógico a la derecha

SAR: Desplazamiento aritmético a la derecha



Las siguientes instrucciones relacionadas ilustran SHR y datos con signo:

INSTRUCCION	AL	COMENTARIO
MOV CL, 03		
MOV AL, 10110111B	; 10110111	
SHR AL, 01	; 11011011	Un corrimiento a la derecha
SHR AL, CL	; 00001011	Tres corrimientos adicionales a la derecha
SHR AX, 03	; Válido para 80186 y procesadores posteriores	

El primer SHR desplaza el contenido de AL un bit hacia la derecha. El bit de mas a la derecha es enviado a la bandera de acarreo, y el bit de mas a la izquierda se llena con un cero. El segundo SHR desplaza tres bits mas al AL. La bandera de acarreo contiene de manera sucesiva 1, 1 y 0; además, tres bits 0 son colocados a la izquierda del AL.

SAR se difiere de SHR en un punto importante: SAR utiliza el bit de signo para llenar el bit vacante de mas a la izquierda. De esta manera, los valores positivos y negativos retienen sus signos. Las siguientes instrucciones relacionadas ilustran SAR y datos con signo en los que el signo es un bit 1:

INSTRUCCION	AL	COMENTARIO
MOV CL, 03		
MOV AL, 10110111B	; 10110111	
SHR AL, 01	; 11011011	Un corrimiento a la derecha
SHR AL, CL	; 00001011	Tres corrimientos adicionales a la derecha
SHR AX, 03	; Válido para 80186 y procesadores posteriores	

En especial, los corrimientos a la derecha son útiles para (dividir entre 2) obtener mitades de valores y son mucho mas rápidas que utilizar una operación de división.

Al terminar una operación de corrimiento, puede utilizar la instrucción JC (Salta si hay acarreo) para examinar el bit desplazado a la bandera de acarreo.

Corrimiento de bits a la izquierda.

Los corrimientos hacia la izquierda (SHL y SAL) mueven los bits a la izquierda, en el registro designado. SHL y SAL son idénticos en su operación. El bit desplazado fuera del registro ingresa a la bandera de acarreo. Las instrucciones de corrimiento hacia la izquierda estipulan datos lógicos (sin signo) y aritméticos (con signo):

SHL: Desplazamiento lógico a la izquierda SAL: Desplazamiento aritmético a la izquierda



Las siguientes instrucciones relacionadas ilustran SHL para datos sin signo:

INSTRUCCION	AL	COMENTARIO
MOV CL, 03		
MOV AL, 10110111B	; 10110111	
SHR AL, 01	; 01101110	Un corrimiento a la izquierda
SHR AL, CL	; 01110000	Tres corrimientos mas
SHR AX, 03	; Válido para 80186 y procesadores posteriores	

El primer SHL desplaza el contenido de AL un bit hacia la izquierda. El bit de mas a la izquierda ahora se encuentra en la bandera de acarreo, y el ultimo bit de la derecha del AL se llena con cero. El segundo SHL desplaza tres bits mas a el AL. La bandera de acarreo contiene en forma sucesiva 0, 1 y 1, y se llena con tres ceros a la derecha del AL.

Los corrimientos a la izquierda llenan con cero el bit de mas a la derecha. Como resultado de esto, SHL y SAL don idénticos. Los corrimientos a la izquierda en especial son útiles para duplicar valores y son mucho mas rápidos que usar una operación de multiplicación.

Al terminar una operación de corrimiento, puede utilizar la instrucción JC (Salta si hay acarreo) para examinar el bit que ingreso a la bandera de acarreo.

- **ROTACION DE BITS (Desplazamiento circular)**

Las instrucciones de rotación, que son parte de la capacidad lógica de la computadora, pueden realizar las siguientes acciones:

1. Hacer referencia a un byte o a una palabra.
2. Hacer referencia a un registro o a memoria.
3. Realizar rotación a la derecha o a la izquierda. El bit que es desplazado fuera llena el espacio vacante en la memoria o registro y también se copia en la bandera de acarreo.
4. Realizar rotación hasta 8 bits en un byte, 16 bits en una palabra y 32 bits en una palabra doble.
5. Realizar rotación lógica (sin signo) o aritmética (con signo).

El segundo operando contiene un valor de rotación, el cual es una constante (un valor inmediato) o una referencia al registro CL. Para los procesadores 8088/8086, la constante inmediata solo puede ser 1; un valor de rotación mayor que 1 debe estar contenido en el registro CL. Procesadores posteriores permiten constantes inmediatas hasta el 31. El formato general para la rotación es:

[etiqueta:] Rotación {registro/memoria}, {CL/inmediato}

Rotación a la derecha de bits

Las rotaciones a la derecha (ROR y RCR) desplazan a la derecha los bits en el registro designado. Las instrucciones de rotación a la derecha estipulan datos lógicos (sin signo) o aritméticos (con signo):

ROR: Rotacion logica a la derecha

RCR: Rotacion a la derecha con acarreo



Las siguientes instrucciones relacionadas ilustran ROR:

INSTRUCCION	BH	COMENTARIO
MOV CL, 03		
MOV BH, 10110111B	; 10110111	
SHR BH, 01	; 11011011	Una rotación a la derecha
SHR BH, CL	; 00001011	Tres rotaciones a la derecha
SHR BX, 03	; Válido para 80186 y procesadores posteriores	

El primer ROR desplaza el bit de mas a la derecha del BH a la posición vacante de mas a la izquierda. La segunda y tercera operaciones ROR realizan la rotación de los tres bits de mas a la derecha.

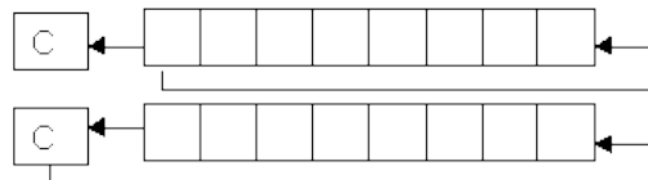
RCR provoca que la bandera de acarreo participe en la rotación. Cada bit que se desplaza fuera de la derecha se mueve al CF y el bit del CF se mueve a la posición vacante de la izquierda.

Rotación a la izquierda de bits

Las rotaciones a la izquierda (ROL y RCL) desplazan a la izquierda los bits del registro designado. Las instrucciones de rotación a la izquierda estipulan datos lógicos (sin signo) y aritméticos (con signo):

ROL: Rotacion logica a la izquierda

RCL: Rotacion a la izquierda con acarreo



Las siguientes instrucciones relacionadas ilustran ROL:

INSTRUCCION	BL	COMENTARIO
MOV CL, 03		
MOV BL, 10110111B	; 10110111	

SHR	BL, 01	; 11011011	Una rotación a la izquierda
SHR	BL, CL	; 00001011	Tres rotaciones a la izquierda
SHR	BX, 03		; Válido para 80186 y procesadores posteriores

El primer ROL desplaza el bit de mas a la izquierda del BL a la posición vacante de mas a la derecha. La segunda y tercera operaciones ROL realizan la rotación de los tres bits de mas a la izquierda.

De manera similar a RCR, RCL también provoca que la bandera de acarreo participe en la rotación. Cada bit que se desplaza fuera por la izquierda se mueve al CF, y el bit del CF se mueve a la posición vacante de la derecha.

Puede usar la instrucción JC (salta si hay acarreo) para comprobar el bit rotado hacia la CF en el extremo de una operación de rotación.

6.5. MULTIPLICACION Y DIVISION.

- MULTIPLICACION**

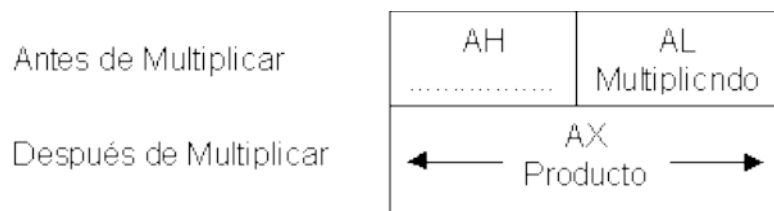
Para la multiplicación, la instrucción MUL maneja datos sin signo y la instrucción IMUL (multiplicación entera) maneja datos con signo. Ambas instrucciones afectan las banderas de acarreo y de desbordamiento. Como programador, usted tiene el control sobre el formato de los datos que procesa, y tiene la responsabilidad de seleccionar la instrucción de multiplicación apropiada. El formato general de MUL e IMUL es :

[etiqueta:] MUL/IMUL registro/memoria

Las operaciones de multiplicación básicas son byte a byte, palabra por palabra y palabras dobles por palabra dobles.

Byte por byte

Para multiplicar dos números de un byte, el multiplicando esta en el registro AL y el multiplicador es un byte en memoria o en otro registro. Para la instrucción MUL DL, la operación multiplica el contenido del AL por el contenido del DL. El producto generado esta en el registro AX. La operación ignora y borra cualquier información que pueda estar en el AH.



Palabra por palabra

Para multiplicar dos números de una palabra, el multiplicando esta en el registro AX y el multiplicador es una palabra en memoria o en otro registro. Para la instrucción MUL DX, la

operación multiplica el contenido del AX por el contenido del DX. El producto generado es una palabra doble que necesita dos registros: la parte de orden alto (mas a la izquierda) en el DX y la parte de orden bajo (mas a la derecha) en el AX. La operación ignora y borra cualquier información que puede estar en el DX.

Antes de Multiplicar	DX	AX
	Ignorado	Multiplicando
Después de Multiplicar	Parte alta de producto	Parte baja del producto

Palabra doble por palabra doble

Para multiplicar dos números de palabras dobles, el multiplicando esta en el registro EAX y el multiplicador es una palabra doble en memoria o en otro registro. El producto es generado en el par EDX:EAX. La operación ignora y borra cualquier información que ya este en el EDX.

Antes de Multiplicar	EDX	EAX
	Ignorado	Multiplicando
Después de Multiplicar	Parte alta de producto	Parte baja del producto

En los ejemplos siguientes, el multiplicador esta en un registro, el cual especifica el tipo de operación:

INSTRUCCION	MULTIPLICADOR	MULTIPLICANDO	PRODUCTO
MUL CL	byte	AL	AX
MUL BX	palabra	AX	DX:AX
MUL EBX	palabra doble	EAX	EDX:EAX

En los ejemplos siguientes, los multiplicadores están definidos en memoria:

```

BYTE1  DB  ?
WORD1  DW  ?
DWORD1 DD  ?

```

OPERACION	MULTIPLICADOR	MULTIPLIANDO	PRODUCTO
-----------	---------------	--------------	----------

MUL BYTE1	BYTE1	AL	AX
MUL WORD1	WORD1	AX	DX:AX
MUL DWORD1	DWORD1	EAX	EDX:EAX

- DIVISION**

Para la división, la instrucción DIV (dividir) maneja datos sin signo y la IDIV (división entera) maneja datos con signo. Usted es responsable de seleccionar la instrucción apropiada. El formato general para DIV/IDIV es:

[etiqueta:] IDIV/DIV {registro/memoria}

Las operaciones de multiplicación básicas son byte entre byte, palabra entre palabra y palabras dobles entre palabra dobles.

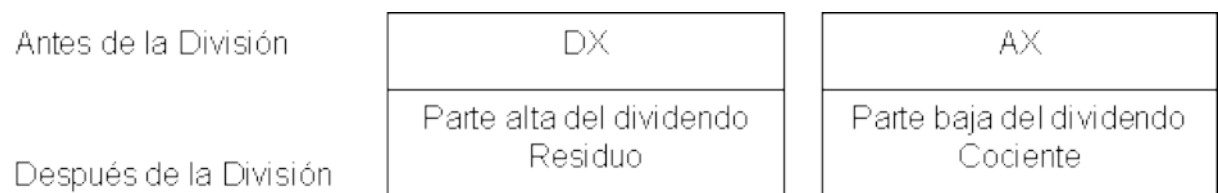
Palabra entre palabra

Aquí el dividendo esta en el AX y el divisor es un byte en memoria o en otro registro. Después de la división, el residuo esta en el AH y el cociente esta en el AL. Ya que un cociente de un byte es muy pequeño -si es sin signo, un máximo de +255 (FFH) y con signo +127 (7FH)- esta operación tiene un uso limitado.



Palabra doble entre palabra

Para esta operación, el dividendo esta en el par DX:AX y el divisor es una palabra en memoria o en otro registro. Después de la división, el residuo esta en el DX y el cociente esta en el AX. El cociente de una palabra permite para datos sin signo un máximo de +32, 767 (FFFFH) y con signo +16, 383 (7FFFH). Tenemos:



Palabra cuádruple entre palabra doble

Al dividir una palabra cuádruple entre una palabra doble, el dividendo esta en el par EDX:EAX y el divisor esta en una palabra doble en memoria o en otro registro. Después de la división, el residuo esta en el EDX y el cociente en el EAX.

Antes de la División	EDX	EAX
Después de la División	Parte alta del dividendo Residuo	Parte baja del dividendo Cociente

En los ejemplos siguientes, de DIV, los divisores están en un registro, que determina el tipo de operación:

OPERACION	DIVISOR	DIVIDENDO	COCIENTE	RESIDUO
DIV CL	byte	AX	AL	AH
DIV CX	palabra	DX:AX	Ax	DX
DIV EBX	palabra doble	EDX:EAX	EAX	EDX

En los ejemplos siguientes de DIV, los divisores están definidos en memoria:

BYTE1	DB	?			
WORD1	DW	?			
DWORD1	DD	?			
...					
	DIVISOR	DIVIDENDO	COCIENTE	RESIDUO	
DIV BYTE1	BYTE1	AX	AL	AH	
DIV WORD1	WORD1	DX:AX	AX	DX	
DIV DWORD1	DWORD1	EDX:EAX	EAX	EDX	

6.6. COMPARACION.

- **LA INSTRUCCION CMP**

La instrucción CMP pro lo común es utilizada para comparar dos campos de datos, uno de los cuales están contenidos en un registro. El formato general para CMP es:

[etiqueta:] CMP {registro/memoria}, {registro/memoria/inmediato}
--

El resultado de una operación CMP afecta la banderas AF, CF, OF, PF, SF y ZF, aunque no tiene que probar estas banderas de forma individual. El código siguiente prueba el registro BX por un valor cero:

```
X CMP BX, 00 ;Compara Bx con cero
JZ B50 ;Si es cero salta aB50
. ;(Acción si es diferente de cero)
.
B50: ... ;Destino del salto, si BX es cero
```

Si el BX tiene cero, cmp establece ZF a 1 y puede o no cambiar la configuración de otras banderas. La instrucción JZ (salta si es cero) solo prueba la bandera ZF. Ya que ZF tiene 1 (que significa una condición cero), JZ transfiere el control (salta) a la dirección indicada por el operando B50.

Observe que la operación compara el primer operando con el segundo; por ejemplo, el valor del primer operando es mayor que, igual o menor que el valor del segundo operando?

- **LA INSTRUCCION CMPS**

CMPS compara el contenido de una localidad de memoria (direccionada por DS:SI). Dependiendo de la bandera de dirección, CMPS incrementa o disminuye también los registros SI y DI en 1 para bytes, en 2 para palabras y en 4 para palabras dobles. La operación establece las banderas AF, CF, OF, PF, SF y ZF.

Cuando se combinan con un prefijo REP y una longitud en el CX, de manera sucesiva CMPS puede comparar cadenas de cualquier longitud. Pero observe que CMPS proporciona una comparación alfanumérica, esto es, una comparación de acuerdo a con los valores ASCII. Considere la comparación de dos cadenas que contienen JEAN y JOAN. Una comparación de izquierda a derecha, tiene el resultado siguiente:

J:J Iguales
E:O Diferentes (E es menor)
A:A Iguales
N:N Iguales

Una comparación de los 4 bytes termina con una comparación de N con N (iguales). Ahora ya que los dos nombres no son idénticos, la operación debe terminar tan pronto como la comparación entre 2 caracteres sea diferente.

Algunas derivaciones de CMPS son las siguientes:

- CMPSB. Compara bytes.
- CMPSD. Compara palabras dobles.
- CMPSW. Compara palabras.

A continuación se muestra la codificación del uso del CMPS y sus derivaciones:

```
TITLE P12CMPST (COM) Uso de CMPS para operaciones en cadenas
.MODEL SMALL
```

```

        .CODE
        ORG     100H
BEGIN: JMP     SHORT MAIN
;-----
NOM1 DB      'Assemblers' ;Elementos de datos
NOM2 DB      'Assemblers'
NOM3 DB      10 DUP ( ' ')
;-----
MAIN  PROC    NEAR          ;Procedimiento principal
        CLD ;Izquierda a derecha
        MOV     CX, 10      ;Iniciar para 10 bytes
        LEA     DI, NOM2
        LEA     SI, NOM1
        REPE    CMPSB       ;Compare NOM1:NOM2
        JNE     G20         ;No es igual, saltarlo
        MOV     BH,01       ;Igual, fijar BH

G20:   MOV     CX, 10      ;Iniciar para 10 bytes
        LEA     DI, NOM3
        LEA     SI, NOM2
        REPE    CMPSB       ;Compare NOM2:NOM3
        JE      G30         ;Igual, salir
        MOV     BL, 02      ;No es igual, fijar BL

G30:   MOV     AX, 4C00H    ;Salir a DOS
        INT     21H
MAIN  ENDP
        END      BEGIN

```

6.7. SALTOS CONDICIONALES E INCONDICIONALES.

Hasta este punto los programas que hemos examinado han sido ejecutados en forma lineal, esto es con una instrucción secuencialmente a continuación de otra. Sin embargo, rara vez un programa programable es tan sencillo. La mayoría de los programas constan de varios ciclos en los que una serie de pasos se repite hasta alcanzar un requisito específico y varias pruebas para determinar que acción se realiza de entre varias posibles.

Requisitos como este implican la transferencia de control a la dirección de una instrucción que no sigue de inmediato de la que se está ejecutando actualmente. Una transferencia de control puede ser hacia adelante, para ejecutar una serie de pasos nuevos, o hacia atrás, para volver a ejecutar los mismos pasos.

Ciertas instrucciones pueden transferir el control fuera del flujo secuencial normal añadiendo un valor de desplazamiento al IP.

Direcciones Corta, cercana y lejana

Una operación de salto alcanza una dirección corta por medio de un desplazamiento de un byte, limitado a una distancia de -128 a 127 bytes. Una operación de salto alcanza una dirección cercana por medio de un desplazamiento de una palabra, limitado a una distancia de -32, 768 a 32, 767 bytes dentro del mismo segmento. Una dirección lejana puede estar en otro

segmento y es alcanzada por medio de una dirección de segmento y un desplazamiento; CALL es la instrucción normal para este propósito.

La tabla siguiente indica las reglas sobre distancias para la operaciones JMP, LOOP y CALL. Hay poca necesidad de memorizar estas reglas, ya que el uso normal de estas instrucciones en rara ocasión causa problemas.

	Corta	Cercana	Lejana
Instrucciones	Mismo Segmento -128 a 127	Mismo Segmento -32,768 a 32,767	Otro Segmento
JMP	Sí	Sí	Sí
Jnnn	Sí	Sí: 80386 y posteriores	No
LOOP	Sí	No	No
CALL	N/A	Sí	Sí

Etiquetas de instrucciones

Las instrucciones JMP, Jnnn (salto condicional) y LOOP requieren un operando que se refiere a la etiqueta de una instrucción. El ejemplo siguiente salta a A90, que es una etiqueta dada a una instrucción MOV:

```
JMP A90
...
A90: MOV AH, 00
...
```

La etiqueta de una instrucción, tal como A90:, terminada con dos puntos (:) para darle atributo de cercana - esto es, la etiqueta esta dentro de un procedimiento en el mismo segmento de código.

Cuidado: Un error común es la omisión de los dos puntos. Note que una etiqueta de dirección en un operando de instrucción (como JMP A90) no tiene un carácter de dos puntos.

La instrucción JMP (Salto incondicional)

Una instrucción usada comúnmente para la transferencia de control es la instrucción JMP (jump, salto, bifurcación). Un salto es incondicional, ya que la operación transfiere el control bajo cualquier circunstancia. También JMP vacía el resultado de la instrucción previamente procesada; por lo que, un programa con muchas operaciones de salto puede perder velocidad de procesamiento. El formato general para JMP es:

[etiqueta] JMP dirección corta, cercana o lejana
--

Una operación JMP dentro del mismo segmento puede ser corta o cercana (o de manera técnica, lejana, si el destino es un procedimiento con el atributo FAR). En su primer paso por un programa fuente, el ensamblador genera la longitud de cada instrucción. Sin embargo, una instrucción JMP puede ser de dos o tres bytes de longitud. Una operación JMP a una etiqueta dentro de -128 a + 127 bytes es un salto corto.

El ensamblador genera un byte para la operación (EB) y un byte para el operando. El operando actúa como un valor de desplazamiento que la computadora suma al registro IP cuando se ejecuta el programa. El ensamblador ya puede haber encontrado el operando designado (un salto hacia atrás) dentro de -128 bytes, como en:

```
A50:
...
JMP A50
```

En este caso, el ensamblador genera una instrucción de máquina de dos bytes. Una JMP que excede -128 a 127 bytes se convierte en un salto cercano, para que el ensamblador genere un código de máquina diferente (E9) y un operando de dos bytes (procesadores 8088/8086) o un operando de cuatro bytes (procesadores 80386 y posteriores). En un salto hacia adelante, el ensamblador aun no ha encontrado el operando designado:

```
JMP A90
...
A90:
```

Ya que algunas versiones del ensamblador no saben en este punto si el salto es corto o cercano, generan de forma automática una instrucción de tres bytes.

	Page 60,132		
TITLE	P08JUMP (COM) Uso de JMP para iterar		
	.MODEL	SMALL	
	.CODE		
	ORG	100H	
MAIN	PROC	NEAR	
	MOV	AX,01	;Iniciación de AX,
	MOV	BX,01	;BX y
	MOV	CX,01	;CX a 01
A20:			
	ADD	AX, 01	;Sumar 01 a AX
	ADD	BX, AX	;Sumar AX a BX
	SHL	CX, 1	;Multiplicar por dos a CX
	JMP	A20	;Saltar a la etiqueta A20
MAIN	ENDP		
	END	MAIN	

La instrucción LOOP

La instrucción LOOP, requiere un valor inicial en el registro CX. En cada iteración, LOOP de forma automática disminuye 1 de CX. Si el valor en el CX es cero, el control pasa a la instrucción que sigue; si el valor en el CX no es cero, el control pasa a la dirección del operando. La distancia debe ser un salto corto, desde -128 hasta +127 bytes. Para una operación que exceda este límite, el ensamblador envía un mensaje como "salto relativo fuera de rango". El formato general de la instrucción LOOP es:

[etiqueta:] LOOP dirección corta

El siguiente programa muestra el funcionamiento de la instrucción LOOP.

```

Page 60,132
TITLE P08LOOP (COM) Ilustración de LOOP
.MODEL SMALL
.CODE
ORG 100H
MAIN PROC NEAR
MOV AX,01 ;Iniciación de AX,
MOV BX,01 ;BX y
MOV CX,01 ;CX a 01
MOV CX,10 ;Iniciar
A20: ;Número de iteraciones
ADD AX, 01 ;Sumar 01 a AX
ADD BX, AX ;Sumar AX a BX
SHL DX, 1 ;Multiplicar por dos a DX
LOOP A20 ;Iterar si es diferente de cero
MOV AX, 4C00H ;Salida a DOS
MAIN ENDP
END MAIN

```

Existen dos variaciones de la instrucción LOOP, ambas también decrementan el CX en 1. LOOPE/LOOPZ (repite el ciclo mientras sea igual o repite el ciclo mientras sea cero) continua el ciclo mientras que el valor en el CX es cero o la condición de cero esta establecida.

LOOPNE/LOOPNZ (repite el ciclo mientras no sea igual o repite el ciclo mientras sea cero) continua el ciclo mientras el valor en el CX no es cero o la condición de cero no esta establecida.

INSTRUCCIONES DE SALTO CONDICIONAL

El ensamblador permite usar una variedad de instrucciones de salto condicional que transfieren el control dependiendo de las configuraciones en el registro de banderas. Por ejemplo, puede comparar dos campos y después saltar de acuerdo con los valores de las banderas que la comparación establece. El formato general para el salto condicional es:

[etiqueta:] Jnnn dirección corta

Como ya se explico la instrucción LOOP disminuye el registro CX; si es diferente de cero, transfiere el control a la dirección del operando. podría reemplazar el enunciado LOOP A20 de la figura anterior con dos enunciados - uno que decremente el CX y otro que realice un salto condicional:

```
DEC CX ;Equivalente a LOOP
JNZ A20
...
```

DEC y JNZ realizan exactamente lo que hace LOOP. DEC decrementa en 1 CX y pone a 1 o a 0 la bandera de cero (ZF) en el registro de banderas. Después JNZ prueba la configuración de la bandera de cero; si el CX es diferente de cero, el control pasa a A20, y si el CX es cero el control pasa a la siguiente instrucción hacia abajo

Datos con signo y sin signo

Distinguir el propósito de los saltos condicionales debe clarificar su uso. El tipo de datos (sin signo o con signo) sobre los que se realizan las comparaciones o la aritmética puede determinar cual es la instrucción a utilizar. Un dato sin signo trata todos los bits como bits de datos; ejemplos típicos son las cadenas de caracteres, tal como nombres o direcciones, y valores numéricos tal como números de cliente. Un dato con signo trata el bit de mas a la izquierda como un signo, en donde 0 es positivo y 1 es negativo.

En el ejemplo siguiente, el AX contiene 11000110 y el BX contiene 00010110. La siguiente instrucción

```
CMP AX, BX
```

compara el contenido de AX con el contenido del BX. Para datos sin signo, el valor AX es mayor; sin embargo, para datos con signo el valor AX es menor a causa del signo negativo.

Saltos con base en datos sin signo

Las instrucciones siguientes de salto condicional se aplican a datos sin signo:

SIMBOLO	DESCRIPCION	BANDERA EXAMINADA
JE / JZ	Salta si es igual o salta si es cero	ZF
JNE / JNZ	Salta si no es igual o salta si no es cero	ZF
JA / JNBE	Bifurca si es mayor o salta si no es menor o igual	CF, ZF
JAE / JNB	Salta si es mayor o igual o salta si no es menor	CF
JB / JNAE	Salta si es menor o salta si no es mayor o igual	CF
JBE / JNA	Salta si es menor o igual o salta si no es mayor	CF, AF

Cada una de estas pruebas las puede expresar en uno de dos códigos simbólicos de operación.

Salto con base en datos con signo

Las instrucciones siguientes de salto condicional se aplican a datos con signo:

SIMBOLO	DESCRIPCION	BANDERA EXAMINADA
JE / JZ	Salta si es igual o salta si es cero	ZF
JNE / JNZ	Salta si no es igual o salta si no es cero	ZF
JG / JNLE	Bifurca si es mayor o salta si no es menor o igual	ZF, SF, OF
JGE / JNL	Salta si es mayor o igual o salta si no es menor	SF, OF
JL / JNGE	Salta si es menor o salta si no es mayor o igual	SF, OF
JLE / JNG	Salta si es menor o igual o salta si no es mayor	ZF, SF, OF

Pruebas aritméticas especiales

Las siguientes instrucciones de salto condicional tienen usos especiales:

SIMBOLO	DESCRIPCION	BANDERA EXAMINADA
JS	Salta si el signo es negativo	SF
JNS	Salta si el signo es positivo	SF
JC	Salta si hay acarreo (igual que JB)	CF
JNC	Salta si no hay acarreo	CF
JO	Salta si hay desbordamiento	OF
JNO	Salta si no hay desbordamiento	OF
JP / JPE	Salta si hay paridad o salta si la paridad es par	PF
JNP / JPO	Salta si no hay paridad o salta si la paridad es impar	PF

No espere memorizar todas estas instrucciones; sin embargo, como recordatorio note que un salto para datos sin signo es igual, superior o inferior, mientras que un salto para datos con signo es igual, mayor que o menor. Los saltos que prueban banderas de acarreo, de desbordamiento y de paridad tienen propósitos únicos.

7.1. DEFINICION.

El segmento de código contiene el código ejecutable de un programa. También tiene uno o mas procedimientos, definidos con la directiva PROC. Un segmento que tiene solo un procedimiento puede aparecer como sigue:

NOMBRE	OPERACION	OPERANDO	COMENTARIO
nomsegmento	SEGMENT	PARA	
nomproc	PROC	FAR	;Un
	.		;procedimiento
	.		;dentro
	.		;del segmento
nomproc	ENDP		;de código
nomsegmento	ENDS		

El nombre del procedimiento debe estar presente, ser único y seguir las reglas para la formación de nombres del lenguaje. El operando far en este caso esta relacionado con la ejecución del programa. Cuando usted solicita la ejecución de un programa, el cargador de programas del DOS utiliza este nombre de procedimiento como el punto de entrada para la primera instrucción a ejecutar.

La directiva ENDP indica el fin de un procedimiento y contiene el mismo nombre que el enunciado PROC para permitir que el ensamblador relacione a los dos. Ya que los procedimientos deben estar por completo dentro de un segmento, ENDP define el final de un procedimiento antes que ENDS defina el final de un segmento.

7.2. LLAMADA DE PROCEDIMIENTOS.

Hasta ahora los segmentos de código han consistido solo en un procedimiento, codificado como:

```
BEGIN PROC FAR
.
.
.
BEGIN ENDP
```

En este caso el operador FAR informa al sistema que la dirección indicada es el punto de entrada para la ejecución del programa, mientras que la directiva ENDP define el final del procedimiento. Sin embargo, un segmento de código puede tener cualquier numero de procedimientos, todos distinguidos por PROC y ENDP. Un procedimiento llamado (o subrutina) es una sección de código que realiza una tarea definida y clara (tal como ubicar el cursor o bien obtener entrada del teclado).

La organización de un programa en procedimientos proporciona los beneficios siguientes:

1. Reduce la cantidad de código, ya que un procedimiento común puede ser llamado desde cualquier lugar en el segmento de código.
2. Fortalece la mejor organización del programa.
3. Facilita la depuración del programa, ya que los errores pueden ser aislados con mayor claridad.

4. Ayuda en el mantenimiento progresivo de programas, ya que los procedimientos son identificados de forma rápida para su modificación.

Operaciones CALL y RET

La instrucción CALL transfiere el control a un procedimiento llamado, y la instrucción RET regresa del procedimiento llamado al procedimiento original que hizo la llamada. RET debe ser la última instrucción en un procedimiento llamado. Los formatos generales para CALL y RET son:

[etiqueta:]	CALL	Procedimiento
[etiqueta:]	RET	[inmediato]

El código objeto particular que CALL y RET generan depende de si la operación implica un procedimiento NEAR (cercano) o un procedimiento FAR (lejano).

Llamada y regreso cercanos. Una llamada (CALL) a un procedimiento dentro del mismo segmento es cercana y realiza lo siguiente:

- Disminuye el SP en 2 (una palabra)
- Mete el IP (que contiene el desplazamiento de la instrucción que sigue al CALL) en la pila.
- Inserta la dirección del desplazamiento del procedimiento llamado en el IP (esta operación vacía el resultado de la instrucción previamente procesada),

Un RET que regresa desde un procedimiento cercano realiza lo siguiente:

- Saca el antiguo valor de IP de la pila y lo envía al IP (lo cual también vacía el resultado de la instrucción previamente procesada).
- Incrementa el SP en 2.

Ahora el CS:IP apunta a la instrucción que sigue al CALL original en la llamada del procedimiento, en donde se reanuda la ejecución.

Llamada y regreso lejanos. Una llamada (CALL) lejana llama a un procedimiento etiquetado con FAR, tal vez en un segmento de código separado. Un CALL lejano mete a la pila al CS y al IP, y RET los saca de la pila.

```
page 60,132
TITLE  P08CALLP (EXE) Llamada a procedimientos
.MODEL  SMALL
.STACK  64
.DATA

;-----
.CODE
BEGIN  PROC    FAR
```

```

CALL B10 ;Llama a B10
;
...
MOV AX,4C00H ;Salida a DOS
INT 21H
BEGIN ENDP
;-----
B10 PROC NEAR
CALL C10 ;Llama a C10
;
...
RET ;De regreso
B10 ENDP ;Quien llama
;-----
END BEGIN

```

7.2. LLAMADA DE PROCEDIMIENTOS.

Hasta ahora los segmentos de código han consistido solo en un procedimiento, codificado como:

```

BEGIN PROC FAR
.
.
.
BEGIN ENDP

```

En este caso el operador FAR informa al sistema que la dirección indicada es el punto de entrada para la ejecución del programa, mientras que la directiva ENDP define el final del procedimiento. Sin embargo, un segmento de código puede tener cualquier número de procedimientos, todos distinguidos por PROC y ENDP. Un procedimiento llamado (o subrutina) es una sección de código que realiza una tarea definida y clara (tal como ubicar el cursor o bien obtener entrada del teclado).

La organización de un programa en procedimientos proporciona los beneficios siguientes:

1. Reduce la cantidad de código, ya que un procedimiento común puede ser llamado desde cualquier lugar en el segmento de código.
2. Fortalece la mejor organización del programa.
3. Facilita la depuración del programa, ya que los errores pueden ser aislados con mayor claridad.
4. Ayuda en el mantenimiento progresivo de programas, ya que los procedimientos son identificados de forma rápida para su modificación.

Operaciones CALL y RET

La instrucción CALL transfiere el control a un procedimiento llamado, y la instrucción RET regresa del procedimiento llamado al procedimiento original que hizo la llamada. RET debe ser la última instrucción en un procedimiento llamado. Los formatos generales para CALL y RET son:

[etiqueta]	CALL	Procedimiento
[etiqueta]	RET	[inmediato]

El código objeto particular que CALL y RET generan depende de si la operación implica un procedimiento NEAR (cercano) o un procedimiento FAR (lejano).

Llamada y regreso cercanos. Una llamada (CALL) a un procedimiento dentro del mismo segmento es cercana y realiza lo siguiente:

- Disminuye el SP en 2 (una palabra)
- Mete el IP (que contiene el desplazamiento de la instrucción que sigue al CALL) en la pila.
- Inserta la dirección del desplazamiento del procedimiento llamado en el IP (esta operación vacía el resultado de la instrucción previamente procesada),

Un RET que regresa desde un procedimiento cercano realiza lo siguiente:

- Saca el antiguo valor de IP de la pila y lo envía al IP (lo cual también vacía el resultado de la instrucción previamente procesada).
- Incrementa el SP en 2.

Ahora el CS:IP apunta a la instrucción que sigue al CALL original en la llamada del procedimiento, en donde se reasume la ejecución.

Llamada y regreso lejanos. Una llamada (CALL) lejana llama a un procedimiento etiquetado con FAR, tal vez en un segmento de código separado. Un CALL lejano mete a la pila al CS y al IP, y RET los saca de la pila.

```
page 60,132
TITLE  P08CALLP (EXE) Llamada a procedimientos
.MODEL  SMALL
.STACK  64
.DATA

;-----
        .CODE
BEGIN  PROC    FAR
        CALL    B10          ;Llama a B10
;
        ...
        MOV     AX,4C00H     ;Salida a DOS
        INT 21H
BEGIN  ENDP
;-----
B10     PROC    NEAR
        CALL    C10          ;Llama a C10
;
        ...
        RET                      ;De regreso
B10     ENDP                ;Quien llama
;-----
        END BEGIN
```

8.2. DEFINICION DE UNA MACRO.

Una definición de macro aparece antes de que cualquier definición de segmento. Examinemos una definición de una macro sencilla que inicializa los registros de segmento para un programa.EXE:

```
INICIAREGS  MACRO           ;Define macro
             MOV AX, @data  ; } Cuerpo de
             MOV DS, AX     ; } la definición
             MOV ES, AX     ; } de la macro
             ENDM           ; Fin de la macro
```

El nombre de esta macro es INICIAREGS, aunque es aceptable cualquier otro nombre valido que sea único. La directiva MACRO en la primer línea indica al ensamblador que las instrucciones que siguen, hasta ENDM ("fin de la macro"), son parte de la definición de la macro. La directiva ENDM termina la definición de la macro.

Los nombres a que se hace referencia en la definición de la macro, @data, AX, DS y ES, deben estar definidos en alguna parte del programa o deben ser dados a conocer de alguna otra forma al ensamblador.

En forma subsecuente se puede usar la macro-instruccion INICIAREGS en el segmento de código en donde quiera inicializar los registros. Cuando el ensamblador encuentra la macro-instruccion INICIAREGS, busca en una tabla de instrucciones simbólicas y, a falta de una entrada, busca macroinstrucciones. Ya que el programa contiene una definición de la macro INICIAREGS, el ensamblador sustituye el cuerpo de la definición generando instrucciones: la expansión de la macro.

Un programa usaría la macroinstruccion INICIAREGS solo una vez, aunque otras macros están diseñadas para ser utilizadas cualquier numero de veces y cada vez el ensamblador genera la misma expansión.

8.3. MANEJO DE PARAMETROS.

Para hacer una macro flexible, puede definir nombres en ella como argumentos mudos (ficticios).La definición de la macro siguiente, llamada DESPLEGAR_MSG, proporciona el uso de la función 09H del DOS para desplegar cualquier mensaje. Cuando se usa la macroinstrucción el programador tiene que proporcionar el nombre del mensaje, el cual hace referencia a un área de datos terminada por un signo de dólar.

```
DESPLEGAR_MSG  MACRO MENSAJE      ; Argumento mudo
                MOV AH, 09H
                LEA DX, MENSAJE
                INT 21H
                ENDM               ; Fin de la macro
```

Un argumento mudo en una definición de macro indica al ensamblador que haga coincidir su nombre con cualquier aparición del mismo nombre en el cuerpo de la macro. Por ejemplo, el argumento mudo MENSAJE también aparece en la instrucción LEA.

Cuando utiliza la macroinstrucción DESPLEGAR_MSG, usted proporciona un parámetro como el nombre real del mensaje que será desplegado, por ejemplo:

DESPLEGAR_MSG MENSAJE2

En este caso, MENSAJE2 tiene que estar apropiadamente definido en el segmento de dato. El parámetro en la microinstrucción corresponde al argumento mudo en la definición original de la macro:

Definición de macro: DESPLEGAR_MSG MACRO MENSAJE (Argumento)
Macroinstruccin: DESPLEGAR_MSG MENSAJE2 (Parametro)

El ensamblador ya ha hecho corresponder el argumento en la definición original de la macro con la instrucción LEA en el cuerpo de la macro. Ahora sustituye el (los) parámetro(s) de la macroinstrucción MENSAJE2 por la presencia de MENSAJE en la instrucción LEA y la sustituye por cualquier otra aparición de MENSAJE.

Un argumento mudo puede contener cualquier nombre valido, incluyendo un nombre de registro tal como CX. Puede definir una macro con cualquier numero de argumentos mudos, separados por coma, hasta la columna 120 de una línea. El ensamblador sustituye los parámetros de la macro instrucción por los argumentos mudos en la definición de la macro, entrada por entrada, de izquierda a derecha.

8.4. MANEJO DE ETIQUETAS LOCALES.

Algunas macros necesitan que se definan elementos de datos y etiquetas de instrucciones dentro de la definición de la macro. Si utiliza la macro mas de una vez en el mismo programa y el ensamblador define los elementos de datos para cada aparición, los nombres duplicados harían que el ensamblador genere un mensaje de error. Para asegurar que cada nombre generado es único, hay que codificar la directiva LOCAL inmediatamente después de la instrucción MACRO. Su formato general es:

LOCAL Etiqueta1, Etiqueta2...Etiquetan.

8.5.BIBLIOTECAS DE MACROS.

Definir una macro y usarla solo una vez en un programa no es muy productivo. El enfoque habitual es catalogar las macros en una biblioteca en disco bajo un nombre descriptivo, como MACRO.LIB. Usted solo tiene que reunir todas las definiciones de sus macros en un archivo y almacenar el archivo en disco:

Macro1 MACRO
....
ENDM

Macro2 MACRO
....
ENDM

Para usar cualquiera de las macros catalogadas, en lugar de codificar las definiciones MACRO

al inicio del programa utilice la directiva INCLUDE así:

```
INCLUDE C: MACRO.LIB  
Macro1  
Macro2
```

El ensamblador accesa el archivo llamado MACRO en la unidad C e incluye ambas definiciones de macro, Macro1 y Macro2, en el programa. El listado ensamblado contendrá una copia de las definiciones de las macros.

La Directiva PURGE.

La ejecución de una instrucción INCLUDE hace que el ensamblador incluya todas las definiciones de macros que están especificadas en la biblioteca. Sin embargo, suponga que una biblioteca contiene las macros SUMA, RESTA, DIVIDE, pero que el programa solo necesita SUMA. La directiva PURGE permite que usted "elimine" las macros RESTA y DIVIDE que no necesita del ensamblado actual:

```
IF1  
INCLUDE C:\MACRO.LIB ;Incluye la biblioteca completa  
ENDIF
```


```
PURGE RESTA, DIVIDE ;Elimina las macros no necesarias
```


Una operación PURGE facilita solo el ensamblado de un programa y no tiene efecto sobre las macros almacenadas en la biblioteca.

BIBLIOGRAFIA

 **Lenguaje Ensamblador y Programacion para PC IBM y Compatibles.**

Peter Abel.
Editorial Prentice Hall.
Tercera Edición

 **Lenguajes Ensambladores**
Fuster Cabañero / Perez Aliaga
Editorial Mc Graw Hill
Primera Edición

 **PC Intern**
Michael Tischer / Bruno Jennrich
Editorial Abacus